ÉCOLE NATIONALE DES CHARTES

Paul, Hector Kervegan

Traitement, exploitation et analyse d'un corpus semi-structuré : le cas des catalogues de vente de manuscrits

Mémoire pour le diplôme de master

Technologies numériques appliquées à l'histoire

2022

Résumé

Introduction

J'ai choisi de structurer mon mémoire autour de plusieurs questions connexes, qui, à différents degrés, se retrouvent tout au long du développement :

En quoi la nature semi-structurée du corpus permet d'en automatiser le traitement ? Comment produire des informations normalisées et exploitables à partir d'un corpus textuel semi-structuré ? En quoi ce traitement et la traduction des documents vers d'autres formats et d'autres médias impacte leur réception ? Quels sont les choix techniques qui influencent cette réception ?

Les deux premières questions, d'orientation plutôt technique, forment la colonne vertébrale pour le mémoire ; elles lient deux aspects centraux : la nature du corpus et la manière dont sa structure permet toute la chaîne de traitement. Par « semi-structuré », j'entends que, à un niveau distant, toutes les entrées de catalogue suivent la même structure ; des séparateurs distinguent les différentes parties, et les informations sont souvent structurées de manière semblable pour chaque manuscrit vendu. Cela permet un traitement de « basse technologie » (*low-tech*) en évitant d'entraîner de lourds modèles de traitement du langage naturel (ce qui aboutirait à des solutions complexes, difficiles à maintenir et à faire évoluer et relativement opaques dans leur fonctionnement). À l'inverse, un corpus semi-structuré peut être traité en déduisant une « structure abstraite », que chaque entrée de catalogue partage. Il est alors possible de mettre en place des solutions techniques plus faciles, pour un résultat de qualité équivalente. Produire des « informations normalisées et exploitables » implique de traiter le corpus en cherchant des réponses à des questions de recherche précises – dans le cadre de mon stage, une question centrale a été de chercher à isoler les facteurs déterminant le prix d'un manuscrit.

Les deux dernières questions, au premier abord plus théoriques, me semblent centrales, notamment à la troisième partie de ce mémoire. Numérisation, traitement informatisé et diffusion sur le web ne sont pas des opérations neutres, mais un ensemble de « traductions » des documents originels. Ces processus comportent une part de choix conscients, qu'il s'agit de mettre en avant. Par exemple, on considère que la majorité des documents vendus ont pour titre l'auteur.ice du document. Cette personne n'est cependant pas toujours mentionnée, et des documents peuvent être nommés d'après un lieu, un évènement ou un thème (la Révolution française, par exemple). Ces « traductions » des catalogues sont relativement discrètes tout au long de la chaîne de traitement (où le

format dominant est la `TEI`, qui garde une relation d'équivalence avec le texte). C'est lors du passage au site web que ce processus de traduction devient plus évident, et, potentiellement, plus problématique. On y abandonne la référence au document originel (les catalogues numérisés ne sont pas accessibles en ligne par un.e utilisateur.ice), le catalogue n'est plus la manière privilégiée d'accéder aux items vendus... De plus, la construction d'un site web implique la conception d'une interface et, dans notre cas, la production d'une série de visualisations intégrées au site. Le passage au site web remet aussi en cause la hiérarchie habituelle entre ingénierie et recherche : la conception d'un site ne répond pas à une question scientifique, mais elle soulève ses propres questions. Loin d'être anodines, ces problématiques de design déterminent la construction et la réception des savoirs. Il est donc important, je pense, de problématiser ces questions de visualisation et de design.

Première partie

Du document numérisé au XML-TEI : nature du corpus, structure des documents et méthode de production des données

Chapitre 1

Le marché des manuscrits autographes au prisme des catalogues de vente

Ce chapitre présente l'objet d'étude du projet *MSS* : étudier le marché des manuscrits autographes du XIX$^{\text{ème}}$ s.. parisien à partir de ses catalogues de vente et étudier la construction du canon littéraire au prisme du marché du manuscrit.

1.1 Pourquoi étudier le marché des manuscrits autographes ?

Cette section porte sur l'intérêt scientifique des objets d'étude du projet (marché des manuscrits et étude de la construction du canon).

1.2 La structure du corpus : périodisation, producteurs des documents et classification

Ici est faite une présentation des documents traités dans le cadre du projet MSS. La présentation est à deux niveaux : au niveau du corpus et des catalogues. Ce chapitre s'appuie sur les mémoires effectués par d'ancien.ne.s stagiaires de *Katabase*, qui ont déjà beaucoup analysé la nature et les enjeux du corpus [1].

1.2.1 Le corpus de catalogues de vente de manuscrits

Ici est présenté le corpus : nature, quantité de documents (et d'entrées individuelles), dates, différentes classifications qui peuvent être faites (revues, catalogues de ventes aux

1. **rondeau__du__noyer__encoder__2019** ; **corbieres__du__2020** ; **janes__du__2021**.

enchères ou à prix fixes...).

1.2.2 Structure des catalogues

Ici sera présentée la structure des catalogues ; la structure de chaque page ne sera détaillée qu'à la partie suivante.

Chapitre 2

Production des données : de l'OCR à la TEI

Cette partie s'attache autant à présenter le processus d'océrisation (qui est déjà bien établi et ne constitue pas le cœur de mon stage) que la structure des documents. Alors que le chapitre précédent s'intéresse aux catalogues dans leur ensemble, ici, on étudie le corpus au niveau de la page et de l'entrée individuelle. En effet, l'océrisation repose sur la segmentation, et donc sur l'établissement d'une structure « abstraite » d'une page (c'est-à-dire, d'un découpage de la page en zones).

2.1 Extraire le texte des imprimés

2.1.1 Comprendre la structure du document pour préparer l'édition numérique

Appréhender la structure de la page à l'aide de SegmOnto

La structure des catalogues est présentée au niveau de la page. L'ontologie SegmOnto [1] est utilisée, autant pour appréhender la structure de la page que pour exprimer cette structure de façon standardisée.

Description des entrées de catalogue : préparer l'édition TEI

Ici, la structure des catalogues est présentée au niveau de l'entrée, c'est à dire du lot mis en vente. C'est à autour de la structure des entrées qu'est construite l'édition XML-TEI. On s'intéresse à la structure des entrées individuelles à deux niveaux :

— Au niveau intellectuel : quelles sont les différentes parties d'une entrée (titre, description du manuscrit, prix...).

1. **christensen_segmonto_2022**.

— Au niveau « textuel » : quels sont les séparateurs, c'est à dire les éléments dans le texte qui permettent de séparer les pages de catalogue en entrées et les entrées en sous-éléments) correspondant à la structure intellectuelle décrite ci-dessus.

2.2 L'encodage des manuscrits en XML-TEI

2.2.1 Encoder les catalogues en TEI

Ici est présentée la représentation XML-TEI des catalogues de vente.

2.2.2 L'encodage en TEI : un processus sélectif qui réduit les significations du texte

Après une étape d'océrisation via *eScriptorium*, le texte extrait des PDF peut être exporté soit en texte brut, soit en XML Page ou Alto. Ces formats s'attachent à garder une relation entre le XML et le document numérisé (les zones de texte sont indiquées, chaque ligne est dans une balise...). Cependant, l'unité intellectuelle centrale à la suite du projet, ce n'est pas la page numérisée, mais l'entrée de catalogue. Un format plus complexe que le XML d'*eScriptorium* est donc nécessaire. Assez logiquement, la suite du projet s'appuie sur une traduction des catalogues en TEI. On s'intéresse autant à la structure des documents XML (quelles balises sont utilisées...) qu'à l'intérêt scientifique d'une édition numérique (balisage sémantique, possibilité de normaliser les informations grâce à des attributs).

L'édition numérique en XML-TEI des catalogues implique une certaine perte d'informations : l'intégralité des significations contenues dans les catalogues imprimés ne peut être traduite en TEI (la police, ou la qualité du papier, peuvent être documentés mais ne peuvent pas être reproduites). Ce genre de perte d'information a lieu, à différents degrés, dans la plupart des éditions TEI : ce format n'est pas un substitut des documents originels. Dans le projet *MSS / Katabase*, d'autres informations sont perdues : l'édition numérique n'est pas censée être une représentation exhaustive des catalogues. La TEI n'est pas utilisée comme un format de conservation, mais comme un format de traitement qui sera enrichi dans les différentes étapes. Afin de mesurer ce qui est conservé et ce qui est perdu du document originel, l'édition TEI sera analysée à la lumière de la « roue du texte » du philologue Patrick Sahle [2] qui modélise les significations plurielles d'un texte.

John Frow + Susan Pearce ?

2. **sahle_digital_2016**.

Deuxième partie

Normalisation, enrichissements et extraction d'informations : une chaîne de traitement pour des données semi-structurées

Chapitre 3

Faire sens d'un corpus complexe : homogénéisation des données et extraction d'informations

3.1 Homogénéiser et normaliser un corpus complexe

Cette section s'intéresse à la manière dont les fichiers `TEI` sont traités afin de pouvoir ensuite en extraire des informations. C'est directement grâce la structure des entrées (et grâce à la nature « semi-structurée » des catalogues) qu'est possible le traitement automatisé des documents.

3.1.1 Pourquoi chercher à normaliser le corpus ?

La question mérite d'être posée : des étapes de post-traitement du corpus sont nécessaires pour pouvoir en extraire des informations et pour pouvoir donc en faire sens ; cependant ce processus peut également impacter la nature du texte et ses significations. Tout comme l'édition `TEI` originelle est un processus sélectif, le traitement des documents encodés est lui un processus sélectif : certaines informations contenues par l'encodage sont privilégiées aux dépens d'autres. Il s'agit ici d'expliciter ces choix (travailler sur les prix, les formats et dimensions des manuscrits plutôt que sur leur sujet) et de les justifier. Cette section s'attache donc à rappeler les questions de recherche qui sous-tendent la normalisation des documents (ajouter plus de structure au document `TEI` pour l'exploiter plus facilement, uniformiser la notation des tailles et des dimensions des documents...).

3.1.2 Comment normaliser le corpus tout en préservant sa valeur documentaire ?

Ici, on s'intéresse à la manière dont la TEI est mise à profit pour enrichir le corpus tout en conservant le contenu textuel des catalogues. Les différentes étapes de normalisation sont également rappelées (ce travail n'étant pas au cœur de mon stage, il s'agira plutôt d'un rappel que d'une présentation technique détaillée).

3.2 Faire sens du corpus : extraction d'informations et fouille de texte

Ici sont décrits le processus et les objectifs de l'extraction d'informations à partir des fichiers TEI. C'est à partir de cette opération d'extraction que sont construites les visualisations, qui permettent une approche graphique du corpus et une meilleure compréhension de celui-ci.

3.2.1 Extraire des informations au niveau des entrées

Des données sont extraites pour chaque entrée de catalogue (un travail largement effectué par A. Bartz, que j'ai légèrement mis à jour) : prix (dans la monnaie de l'époque et en francs constants), date de vente normalisée, nom de l'auteur.ice et description du manuscrit... L'extraction d'informations pour les entrées individuelles permet surtout de faire une réconciliation des manuscrits vendus (c'est à dire, de retrouver les items vendus plusieurs fois).

3.2.2 Extraire des informations au niveau des catalogues

Un second processus d'extraction produit des données pour chaque catalogue de vente : titre du catalogue, date de vente, nombre d'items vendus, prix minimum, inférieur et maximum, prix moyen et médian, variance... Ce processus met l'accent sur une approche statistique et économique du corpus, qui permettra d'étudier l'évolution du cours et du volume du marché des manuscrits (nombre d'items en vente, évolution des prix).

3.2.3 Vers une approche économique du corpus : la conversion automatique des prix en francs constants

En même temps que ces processus d'extraction d'informations, un script de conversion des monnaies (françaises et étrangères) en francs constants 1900 a été élaboré. En annulant l'effet de l'inflation, les francs constant permettent d'étudier l'évolution réelle des prix.

Chapitre 4

Vers une étude des facteurs déterminant le prix des documents : alignement des entrées du catalogue avec *Wikidata* et exploitation de données normalisées

Ce chapitre est construit autour d'une question de recherche : comment produire des informations exploitables pour une étude économétrique à partir d'un corpus textuel semi-structuré ? Un des objectifs du projet est de faire l'étude des facteurs déterminant le prix d'un manuscrit. Pour faire cette étude, il faut obtenir, pour chaque entrée du catalogue, un certain nombre d'informations normalisées. Le travail d'extraction de données présentes dans les catalogues a déjà été fait par de précédent.e.s stagiaires. Ces données sont principalement quantitatives : prix des manuscrits, dimensions et nombre de pages, date de création. Il est nécessaire de compléter les informations par des données qualitatives et d'enrichir les données disponibles avec des sources extérieures. Pour ce faire, il a été choisi d'aligner le nom des auteur.ice.s des manuscrits avec des identifiants *Wikidata* ; dès lors que l'on a un identifiant *Wikidata*, il est possible de récupérer automatiquement des informations sur les personnes via `SPARQL`. Le choix de travailler uniquement sur les noms, et non sur la description des documents, a deux motivations :

— Les noms de personnes (et la manière dont elles sont décrites) constituent la partie la plus normalisée des documents. La description des manuscrits est plutôt en « texte libre ». Dans la continuité avec le reste du projet, nous sommes resté dans une approche « basse technologie », qui consiste à s'appuyer majoritairement sur des solutions techniquement simples. C'est pourquoi nous avons préférer traiter les noms

avec des tables de correspondance [1] et des expressions régulières, plutôt que de faire du TAL sur la description des documents.

— Toutes les informations « simples » (données quantitatives facilement normalisables : dates etc.) ont déjà été extraites des descriptions des manuscrits.

Ce travail d'enrichissement a été fait en deux temps.

La première étape, et la plus difficile, est l'alignement avec *Wikidata*. Cela demande d'extraire un ensemble d'informations à partir du nom de la personne et de la description de celle-ci. Parmi les informations extraites : nom, prénom, titre de noblesse, occupation, dates de vie et de mort. À partir de ces informations, stockées dans un dictionnaire, un algorithme construit successivement différentes chaînes de caractères pour lancer des recherchers en plein texte sur l'`Interface de programmation d'application (API)` de *Wikidata*. L'objectif est que le premier résultat recherché sur *Wikidata* soit correct. Sur un jeu de test, le score F1 obtenu est de 68%. Un relecture « manuelle » des résultats est donc nécessaire.

La deuxième étape, nettement plus simple, consiste à lancer des requêtes *Wikidata* sur les identifiants récupérés afin de récupérer des informations sur les auteur.ice.s des manuscrits (cette partie du travail est encore en cours) pour enrichir nos données.

Une fois ce travail effectué, l'enrichissement des données à proprement parler est possible : les fichiers `TEI` sont mis à jour pour ajouter les identifiants *Wikidata*. Ainsi, il est possible de faire le lien entre les entrées de catalogues dans des fichiers `XML` et les données issues de requêtes `SPARQL`, stockées dans un `JSON`.

4.1 Questions introductives : pourquoi et comment s'aligner avec *Wikidata* ?

Cette section, introductive, répond à des questions évidentes mais essentielles : elles permettent de mettre au clair l'intérêt et les (multiples) difficultés dans l'alignement avec *Wikidata*.

4.1.1 Pourquoi s'aligner avec des identifiants *Wikidata* ?

L'alignement avec *Wikidata* a pour objectif principal de mieux comprendre les déterminants du prix des manuscrits sur le marché du XIX$^{\text{ème}}$ s.. Mais pourquoi passer par un alignement avec *Wikidata* ?

Pour étudier les déterminants du prix d'un manuscrit, il faut établir la relation entre la variable dont la valeur est étudiée (le prix d'un manuscrit) et un ensemble d'autres fac-

1. C'est à dire, des tables qui permettent de normaliser la manière dont les informations figurent dans les catalogues, et donc de remplacer des termes « vernaculaires » par leurs équivalents utilisés par *Wikidata*

teurs (qui a écrit un manuscrit, quelles sont ses dimensions, de quand date le document...). En d'autres termes, il faut étudier le comportement d'une variable en fonction d'autres variables. En économétrie, cette opération s'appelle le calcul de régressions linéaires. La variable étudiée (le prix) est dite la variable expliquée ; les facteurs déterminant la valeur de cette variable sont dites « variables explicatives » [2]. Cependant, cette opération est loin d'être anodine : il faut d'abord identifier les variables pertinentes, et ensuite trouver un moyen de les quantifier. Deux difficultés se présentent pour alors.

Premièrement, il faut pouvoir quantifier les variables expliquées pour calculer des régressions linéaires. Il est possible de leur assigner une valeur numéraire (ce qui est aisé pour les informations quantitatives des catalogues : la date de l'écriture d'un manuscrit, ses les dimensions). Une autre possibilité est de quantifier la présence ou non d'une variable : mention d'un.e destinataire ou du contenu d'un manuscrit. Cependant, ces approches quantitatives ne permettent pas de quantifier des informations complexes, comme la célébrité des auteur.ice.s, ou encore si un manuscrit porte sur un évènement historique ou biographique important (le manuscrit d'un texte célèbre, par exemple, pourrait avoir une valeur particulières). Ces informations sont parfois être présentes dans les catalogues ; elles peuvent aussi être connues des lecteur.ice.s d'aujourd'hui et des acheteurs et acheteuses de l'époque. Il n'existe cependant pas de méthodes faciles pour détecter ou quantifier la célébrité d'une personne, ou l'importance d'un sujet.

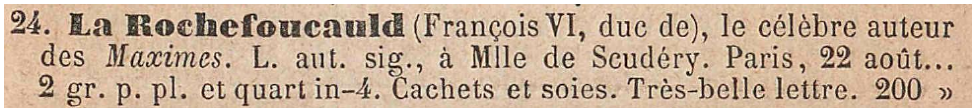
Une deuxième difficulté découle justement de la part d'implicite qu'il y a dans les catalogues. Les descriptions des items vendus sont brèves, et comprendre ce qui fait la valeur d'un manuscrit demande aux acheteur.euse.s d'avoir des références culturelles et historiques : celles-ci permettent d'identifier l'auteur.ice ou le sujet, et donc pour comprendre la valeur d'un manuscrit. Dans le cadre du projet *MSS / Katabase*, les entrées de catalogues sont traitées par une machine qui, en toute logique, ne dispose pas de ces références. La compréhension qualitative des entrées de catalogues n'est donc pas compatible avec l'approche par lecture distante du projet. Pour éviter de perdre ces informations qualitatives essentielles, il est donc nécessaire de trouver un moyen de quantifier le qualitatif.

En bref, la question est : comment faire la différence entre une lettre de La Rochefoucauld (4.1a), vendue 200 francs, et la deuxième (4.1b), vendue à 30 francs ? Le problème est un problème de lecture. Une observation de la description des lettres par un être humain comme par une machine peuvent identifier des éléments semblables dans le texte : les deux lettres sont écrites par des ducs ; l'une est une « Très-belle lettre » (4.1a), l'autre est une « Lettre intéressante » (4.1b) [3]. Bien que les lettres partagent des attributs, il y a une forte différence de prix entre les deux manuscrits. Un.e lecteur.ice peut trouver une raison à cette différence de prix : La Rochefoucauld et Madeleine de Scudéry n'ont pas

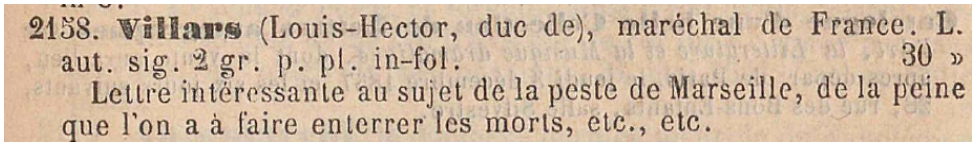
2. **noauthor__regression__2022**.

3. Ce sont des informations qui se retrouvent souvent, et il est donc possible d'écrire un programme qui les relève automatiquement

le même statut que le duc de Villars. Un regard humain peut donc interpréter un prix et déterminer une valeur en s'appuyant sur ses connaissances. La lecture est qualitative et s'appuie sur de l'implicite, ce qui n'est pas possible pour une machine : formellement, rien ne distingue un nom d'un autre ; lorsqu'un programme « lit » un texte, il ne peut pas s'appuyer sur ses connaissances pour déterminer ce qu'un nom signifie, ce à quoi il fait référence.


24. **La Rochefoucauld** (François VI, duc de), le célèbre auteur des *Maximes*. L. aut. sig., à Mlle de Scudéry. Paris, 22 août... 2 gr. p. pl. et quart in-4. Cachets et soies. Très-belle lettre. 200 »

(a) Une lettre écrite par La Rochefoucauld vendue à 200 francs.


2158. **Villars** (Louis-Hector, duc de), maréchal de France. L. aut. sig. 2 gr. p. pl. in-fol. 30 »
Lettre intéressante au sujet de la peste de Marseille, de la peine que l'on a à faire enterrer les morts, etc., etc.

(b) Une lettre écrite par Louis-Hector Villars vendue à 30 francs.

FIGURE 4.1 – Deux exemples de lettres

Pour analyser efficacement la variable « prix », il faut pourtant pouvoir, dans une certaine mesure, comprendre les informations implicites et qualitatives contenues dans les catalogues. Le parti pris a donc été de construire le socle de connaissance qui manque à une machine, en s'alignant avec *Wikidata* et en s'en servant pour enrichir nos données. Le choix a été fait de ne s'aligner avec *Wikidata* que pour certaines parties des entrées de catalogue. Pour rappel, voici leur structure (4.1) :

Les entrées de catalogue contiennent beaucoup d'informations qualitatives, qui pourraient avoir une influence sur le prix du manuscrit : ici par exemple, la description du contenu de la lettre dans le `note` ; il est également souvent fait mention du ou de la destinataire. Cependant, l'alignement avec *Wikidata* n'a pas été fait avec l'intégralité des entrées. C'est seulement le contenu du `name` qui a été aligné avec *Wikidata*, à l'aide des informations contenues dans le `trait`. Le `desc` a déjà fait l'objet d'un grand travail de normalisation et d'extraction d'informations ; un alignement avec des sources externes n'aurait donc pas une très grande plus-value. L'élément `note` contient souvent des informations intéressantes, puisque c'est là qu'est décrit le contenu d'un manuscrit. Cependant, cet élément n'est pas toujours présent ; son contenu est souvent écrit en langage naturel, non structuré, et contient des informations trop variées pour développer un traitement uniforme. Il est donc difficile de tirer parti de cet élément. Le `name` et son `trait` sont les éléments les plus régulièrement présents ; les informations qu'ils contiennent sont toujours les mêmes (nom d'une personne ou thème d'un manuscrit dans le `name`, description du `name` dans le `trait`) ; enfin, ces deux éléments n'ont pas du tout été transformés dans le reste de la chaîne de traitement. Ils portent donc des informations qualitatives centrales pour produire des données exploitables dans une étude économétrique.

```xml
<item n="287" xml:id="CAT_000126_e287">
    <num type="lot">287</num>
    <name type="author">Tascher de la Pagerie
    ↪ (Marie-Euphémie-Désirée)</name>
    <trait>
        <p>fille de Joseph Tascher qui passa à Saint-Domingue, et de
        ↪ Mlle de la Chevalerie; elle épousa M. de Renaudin, puis le
        ↪ marquis François de Beauharnais et contribua beaucoup à la
        ↪ fortune de la future impératrice Joséphine.</p>
    </trait>
    <desc xml:id="CAT_000126_e287_d1">
        Pièce originale;
        (<date when="1780">1780</date>),
        <measure type="length" unit="p" n="4">4 p.</measure>
        <measure type="format" unit="f"
        ↪ ana="#document_format_4">in-4.</measure>
    </desc>
    <measure commodity="currency" unit="FRF" quantity="15">15</measure>
    <note>Plaidoirie pour le dit Renaudin contre sa femme qu'on accusait
    ↪ d'avoir usé de l'influence de son parent, M. de Beauharnais,
    ↪ pour se faire épouser</note>
</item>
```

Code source 4.1 – Représentation XML-TEI d'une entrée de catalogue

Le parti pris a donc été d'aligner avec des identifiants *Wikidata* les noms contenus dans les balises `name` à l'aide des descriptions contenues dans les `trait` ; à partir de cet alignement a été constituée une base de données. Cela permet d'approximer une lecture « humaine » des items en vente : pour chaque auteur.ice, un certain nombre d'informations auront été récupérées pour mieux identifier la personne (ses occupations, son origine, ses dates de vie...). L'analyse du corpus s'appuit alors sur un bagage de connaissances qui permet d'appréhender par lecture distante l'importance d'une personne. Il devient alors envisageable de voir dans quelle mesure la mention d'une personne impacte le prix d'un manuscrit, et quels sont les facteurs biographiques déterminant dans l'établissement de la valeur. Pour revenir à l'exemple de La Rochefoucauld : à défaut de permettre de savoir qui il est, un alignement avec *Wikidata* permet d'identifier son statut et sa place dans la culture française, en récupérant le nombre de ses publications ou encore les institutions dont il est membre.

4.1.2 Quelle relation avec le liage d'entités nommées ?

blablabla à compléter plus tard

4.1.3 Présentation générale de l'algorithme

Construire un jeu de données issu de `SPARQL` à partir de la manière dont une personne est nommée et décrite au XIX$^{\text{ème}}$ s. n'est pas une opération anodine. La chaîne de traitement est donc assez complexe, comme le montre le schéma 4.2. Cette chaîne de traitement peut être séparée en trois étapes.

Étape 1 – Extraction et structuration de données

Premièrement, il s'agit d'aligner les entrées de catalogue avec des identifiants *Wikidata*. Ceux-ci sont liés à des « entités » *Wikidata* : des personnes, lieux et évènements décrits dans *Wikidata* par un certain nombre de propriétés (date de naissance, lieu de résidences...). Cette première étape repose avant tout sur l'extraction et la traduction des données depuis les éléments `name` et `trait`. Ce processus d'extraction permet de récupérer toutes les données pertinentes pour chaque entrée de catalogue et de les stocker dans un dictionnaire structuré. Comme on le verra, la nature « semi-structurée » des entrées (ainsi qu'une bonne connaissance du corpus) permet de d'automatiser le processus d'extraction et de traduction des données par détection de motifs, sans avoir à passer par l'apprentissage machine : étant donné que les mêmes types d'informations sont toujours présentes et que les entrées suivent des modèles relativement proches, il est possible de s'appuyer sur la structure des entrées pour identifier les informations pertinentes. L'extraction de données repose donc sur de la détection de motifs à l'aide d'expressions régulières : des

récurrences sont repérées dans le texte et utilisées pour distinguer différentes informations (nom, prénom, titre de noblesse...). Pour appuyer l'usage d'expressions régulières par une méthode plus « qualitative » et précise, certains termes particuliers sont extraits et éventuellement traduits à l'aide de tables de conversion (c'est-à-dire de dictionnaires qui associent à un terme dans le texte une version normalisée).

Étape 2 – récupération d'identifiants *Wikidata* via des recherches en plein texte à l'aide d'une `API`

Une fois les données du `name` et du `trait` structurées en dictionnaire, elles sont utilisées pour lancer plusieurs recherches en plein texte sur le moteur de recherche de *Wikidata*. Ces recherches sont faites automatiquement grâce à l'`API` de *Wikidata*. Pour maximiser les chances d'obtenir un identifiant valide, un algorithme a été conçu pour lancer plusieurs recherches à partir de chaque dictionnaire. La première recherche met bout-à-bout toutes les valeurs disponibles dans le dictionnaire. Ensuite, en fonction des paramètres de recherche disponibles dans le dictionnaire, différentes autres recherches sont lancées. Cet algorithme a été élaboré en menant de nombreux tests pour maximiser le taux de réussite, calculé sous la forme d'un score F1.

Étape 3 – constitution d'un jeu de données à l'aide de `SPARQL`

Si l'étape d'alignement avec *Wikidata* est la plus complexe, elle n'est qu'une étape préparatoire vers la constitution du jeu de données. En fait, récupérer les identifiants est seulement ce qui rend possible l'enrichissement en tant que tel : en lançant une requête `SPARQL` sur tous ces identifiants, il est possible, pour chaque entité représentée par l'identifiant, de récupérer des informations depuis *Wikidata* et donc de construire le jeu de données définitif. Pour cette étape, le processus est plus simple : les identifiants récupérés à la fin de l'étape précédente sont dédoublonnés (pour éviter de lancer plusieurs fois la même requête) ; ensuite une requête `SPARQL` est initalisée et lancée chacun des identifiants. Les résultats sont traduits depuis les formats `JSON` ou `XML` retournés par `SPARQL` sous forme de `JSON` plus simple, et donc plus aisément manipulable. Le jeu de données est enregistré dans un fichier. Pour finir, les identifiants *Wikidata* sont réinjectés aux catalogues `TEI`, afin de pouvoir faire le lien entre les catalogues et le jeu de données qui a été construit.

Cette chaîne de traitement étant lancée sur plus de 80000 entrées de catalogues, le temps d'exécution est très long et même des petites améliorations de performance peuvent avoir un grand impact ; dans sa version initiale, le script demandait des performances particulièrement élevées, et ne fonctionnait pas sur un ordinateur aux capacités limitées. La chaîne de traitement a donc été reprise en plusieurs points afin d'être optimisée, de fonctionner plus vite en étant moins coûteuse en ressources.

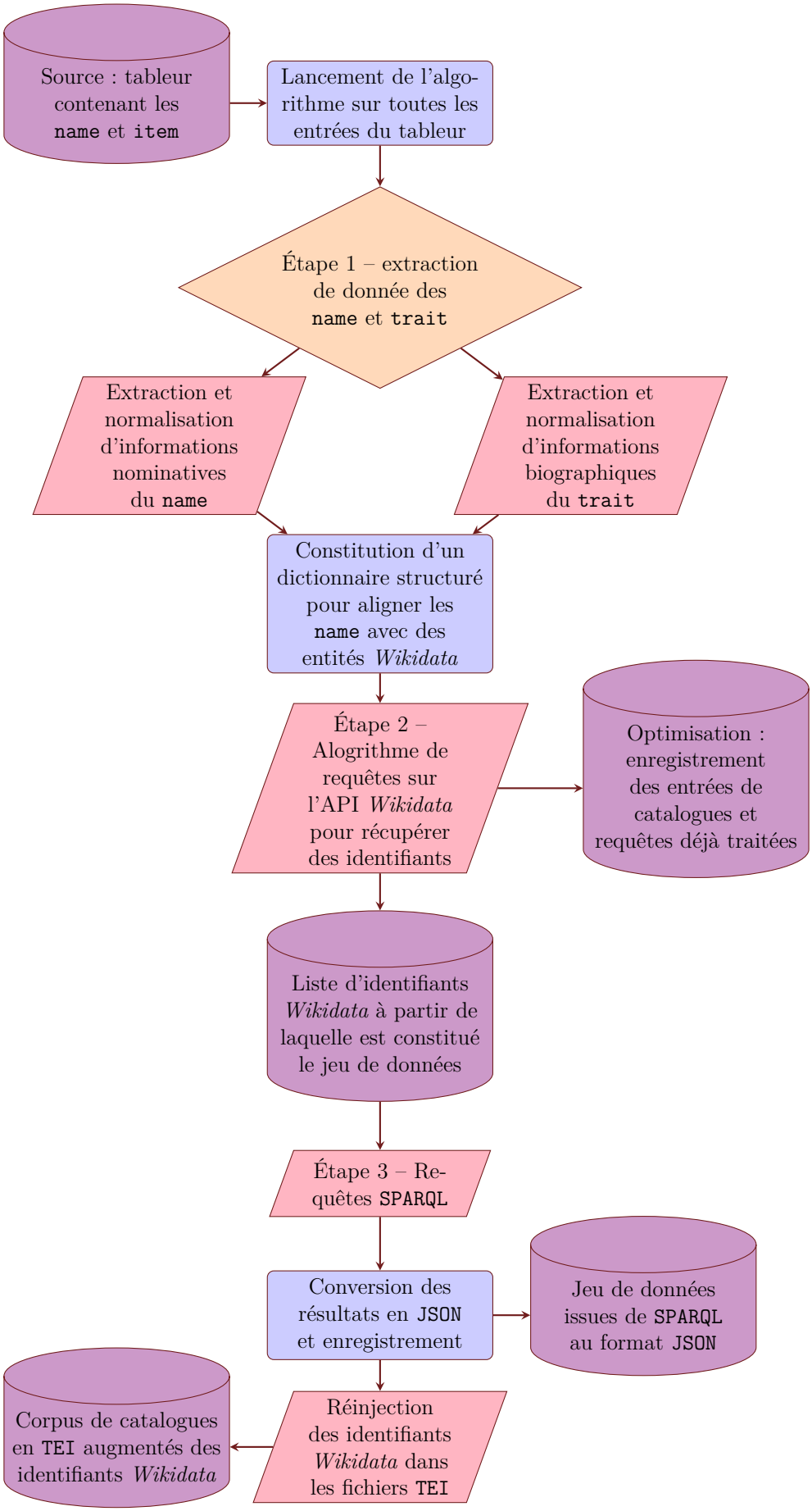


FIGURE 4.2 –
Présentation générale de l'algorithme d'enrichissement de données à l'aide de *Wikidata*

4.1.4 Comment traduire des descriptions textuelles datant du XIX^{ème} s. en chaînes de caractères qui puissent retourner un résultat sur *Wikidata* ?

Dans la réalisation de cet algorithme, la principale difficulté porte sur la récupération d'identifiants *Wikidata* à l'aide de recherches en plein texte. Le script prend en entrée un nom et sa description – tels qu'elles figurent dans des catalogues datant majoritairement du XIX^{ème} s.. La difficulté, au delà de la détection et de l'extraction d'informations, est de traduire ces informations pour qu'elles permettent de trouver des résultats pertinents sur *Wikidata*. Ce problème est autant linguistique de technique. Une personne ou une chose est nommée ou décrite d'une certaine manière dans un catalogue de vente ancien. Il n'y a aucune garantie que cette caractérisation corresponde à celle faite par *Wikidata* : l'orthographe des noms évoluent, tout comme la manière de nommer certains métiers. À ces évolutions orthographiques s'ajoutent des évolutions intellectuelles : les titres de noblesse sont un marqueur plus important au XIX^{ème} s. français que dans un XXI^{ème} s. mondialisé. Une personne n'est que rarement décrite par son titre dans *Wikidata*.

Le problème de la traduction des noms

Il existe bien sûr des cas simples, comme l'exemple 4.2 : en extrayant le contenu du `name`et en traduisant le « roi » issu du `trait`, la chaîne de caractère obtenue est « Henri IV king ». En recherchant cette chaîne de caractère sur *Wikidata*, le premier résultat obtenu est correct. Cependant, de nombreux cas sont plus complexes, surtout lorsque l'auteur.ice du manuscrit est moins célèbre. L'exemple 4.3 est éclairant : dans le catalogue, la personne est nomée « Bruno Daru » ; sur *Wikidata*, le nom de la personne est « Pierre Daru », et son nom complet Pierre Antoine Noël Bruno Daru. Si la recherche en plein texte est faite avec les mêmes paramètres que pour l'exemple précédent (nom de la personne et titre de noblesse), le premier résultat obtenu n'est pas le bon : c'est un renvoi à un article de *l'Encyclopedia Britannica* datant de 1911. C'est en cherchant seulement le nom est le prénom que *Wikidata* retourne un résultat pertinent. Il est intéressant de retenir deux choses de cet exemple : dans les catalogues, le prénom d'une personne correspond en fait souvent à son deuxième ou troisième prénom ; ensuite, le titre de noblesse est un critère plus fréquemment mentionné dans les catalogues que dans *Wikidata*. Cela s'explique assez aisément : le XIX^{ème} s. connaît une alternance de régimes politiques (royauté, empire, république) où la noblesse n'a pas encore perdu son pouvoir. La probabilité qu'un titre de noblesse soit mentionné sur *Wikidata* diminue lorsqu'un titre est peu important ; dans les catalogues, cependant, même les titres les moins importants sont régulièrement mentionnés. Par conséquent, seuls les titres les plus importants seront extraits pour lancer une recherche sur l'`API` de *Wikidata*.

Dans le cas de noms de personnes étrangères, la situation peut être plus complexe

```xml
<item n="134" xml:id="CAT_000233_e134">
    <!-- ... -->
    <name type="author">Henri IV</name>
    <trait>
        <p>roi de France.</p>
    </trait>
    <!-- ... -->
</item>
```

Code source 4.2 – Un cas simple : Henri IV roi de France

```xml
<item n="98" xml:id="CAT_000082_e98">
    <!-- ... -->
    <name type="author">Daru (Bruno, comte)</name>
    <trait>
        <p>célèbre ministre de Napoléon Ier, historien de Venise, de
        ↪ l'Acad. fr., né à Montpellier</p>
    </trait>
    <!-- ... -->
</item>
```

Code source 4.3 – Un cas plus complexe : Pierre Antoine Noël Bruno Daru

encore. L'exemple 4.4 combine différentes difficultés.

— D'abord, la personne est étrangère ; dans les catalogues, les noms sont systématiquement francisés – « Albert-Venceslas-Eusèbe » dans le catalogue, « Albrecht Wenzel Eusebius » en langue originelle. Se pose donc la question de si le nom doit être traduit, et si oui comment ?

— Ensuite, comme l'indique la présence de « dit » dans le `name`, il est mentionné un nom de naissance (« de Waldstein ») et un nom d'usage (« Wallenstein »). Idéalement, il faudrait choisir entre l'un ou l'autre, plutôt que de rechercher « Waldstein Wallenstein » sur *Wikidata*, ce qui risque d'augmenter le bruit.

Notre approche s'appuyant sur la structure du texte, le deuxième point peut être réglé : le nom d'usage est écrit au début, et le nom de naissance entre parenthèses (c'est également le cas des noms de personnes nobles, par exemple). Il est donc possible de choisir l'un ou l'autre nom. Le premier point est plus problématique : si la traduction du nom serait envisageable en théorie, celle-ci est difficilement compatible avec une approche basée sur la détection de motifs dans le texte : le prénom est repérable comme étant un motif (trois noms séparés par des tirets) ; cependant, il est impossible de le traduire automatiquement (ce qui demanderait de connaître la langue dans laquelle un prénom doit être traduit). C'est ici que les informations contenues dans le `trait` prennent leur

```xml
<item n="5518" xml:id="CAT_000401_e5518">
    <!-- ... -->
    <name type="author">Wallenstein (Albert-Venceslas-Eusèbe de
    ↪ Waldstein dit)</name>
    <trait>
        <p>duc de Friedland, célèbre général de la guerre de Trente ans.
        ↪ Assassiné en 1634.</p>
    </trait>
    <!-- ... -->
</item>
```

Code source 4.4 – Le problème des noms de personnes étrangères

importance : lorsqu'il y a des défaillances dans les informations nominatives, des données biographiques permettent de diminuer le risque d'erreurs. Dans cet exemple, recherche « Albert-Venceslas-Eusèbe Waldstein » ne retourne aucun résultat, de même que rechercher Albert-Venceslas-Eusèbe Wallenstein. Cependant, le bon résultat est obtenu en recherchant « Wallenstein 1634 ». Une difficulté supplémentaire vient avec ce type de cas : différents paramètres de recherche (nom, prénom...) ont un impact différent dans l'obtention du bon résultat en fonction des personnes sur qui la requête est faite. Dans ce cas, rechercher le nom d'usage et la date de naissance retourne un résultat valide, ce qui n'est pas toujours le cas. Pour contourner ce problème, trois solutions ont été mises en place : d'abord, ce types de requêtes a été fait « à la main », de façon non-automatique, pour de nombreuses entrées différentes afin de déterminer la meilleure combinaison de caractères ; ensuite, des tests qui permettent de mesurer l'influence de chaque paramètre de recherche dans l'obtention du résultat ; enfin, l'algorithme final lance successivement différentes requêtes avec différents paramètres afin de maximiser la probabilité d'obtenir un résultat valide. Nous reviendrons plus en détail sur les deux derniers points.

L'extraction d'informations biographiques : une autre difficulté

Cependant, le problème ne s'arrête pas qu'aux noms. Dans un exemple ; précédent, le titre de noblesse influençait l'obtention d'un résultat valide. De nombreuses autres informations biographiques pourraient, au premier abord, permettre d'obtenir le bon résultat. C'est souvent le cas, puisque extraire le métier ou la fonction d'une personne permet de supprimer les faux positifs retournés par l'`API`. C'est par exemple le cas dans l'exemple 4.5. En cherchant uniquement le nom et le prénom (« Hans Bulow »), le premier résultat retourné renvoie à un journaliste suédois. Extraire le mot « pianiste » `trait` et le traduit en anglais permet d'obtenir le bon résultat.

L'extraction d'informations biographiques et leur utilisation dans des requêtes est donc pertinent. Cependant, extraire trop d'informations conduit à lancer des requêtes qui

```xml
<item n="136" xml:id="CAT_000189_e136">
    <!-- .. -->
    <name type="author">Bulow (Hans)</name>
    <trait>
        <p>le célèbre pianiste.</p>
    </trait>
</item>
```

Code source 4.5 – Un exemple où l'extraction du métier permet l'obtention du bon résultat

ne renvoient aucun résultat. Dans les exemples 4.6 et 4.7, extraire et traduire des fonctions conduit à lancer les requêtes « John Okey colonel » et « Jean Bouhier president » qui ne retournent aucun résultat, ou des résultats qui ne sont pas valides. Cependant, dans les deux cas, si une requête est lancée sans la fonction, un résultat correct est obtenu. Les raisons pour lesquelles des résultats erronés sont retournés ne sont cependant pas les mêmes, et il est intéressant de mieux observer les requêtes lancées et les résultats obtenus. Dans le premier cas, le terme mis en avant dans le `trait` (« colonel ») n'est pas celui avec lequel la personne est décrite sur *Wikidata* (où John Okey est décrit comme étant un homme politique). Cela met en avant un problème relatif au changement de regard sur des personnalités : dans un contexte, la personne est décrite comme une figure militaire, dans l'autre comme une figure politique. Le deuxième cas est plus technique. Il y a en fait une erreur dans la requête qui ne retourne pas de résultat (« Jean Bouhier president ») : un.e président.e de parlement n'est en général pas décrite comme « président ». Cependant, en extrayant des données uniquement par détection de motifs, il est possible de repérer et traduire un terme générique comme « président ». Extraire le complément « Parlement de Dijon » du `trait` n'est cependant pas possible (cela impliquerait d'étudier la grammaire de la phrase, pour mettre en avant la relation entre « président » et « Parlement de Dijon »). Au vu de la taille et de la variété du jeu de données, il est impossible de traiter au cas par cas des entrées, ou préciser la détection de motif avec suffisamment de précision pour pouvoir résoudre ce genre de difficultés.

De situations comme les exemples 4.6 et 4.7, il faut donc retenir que l'extraction d'informations vient nécessairement avec un risque d'erreur. Le parti pris a donc été de ne pas repérer les métiers et autres termes très spécifiques, comme les grades militaires et les titres de noblesse peu élevés : ils ne retournent pas de résultats sur le moteur de recherche. Ensuite, plus des requêtes sont précises, plus elles risquent de retourner du silence (c'est-à-dire, de ne pas donner de réponse) ; cependant, si un résultat est obtenu, il est plus probable que ce résultat soit correct. Une fois l'extraction d'informations faite, l'algorithme d'extraction d'identifiants sur l'`API` *Wikidata* a donc été conçu en suivant un principe soustractif : les premières recherches sont faites avec un maximum de paramètres ; si aucun résultat n'est obtenu, des paramètres sont enlevés pour que l'`API` retourne un

```xml
<item n="152" xml:id="CAT_000189_e152">
    <!-- ... -->
    <name type="author">Okey (John)</name>
    <trait>
        <p>colonel anglais, un des lieutenants de Cromwell.</p>
    </trait>
    <!-- ... -->
</item>
```

Code source 4.6 – Quand l'extraction d'un métier conduit à des requêtes trop spécifiques

```xml
<item n="5430" xml:id="CAT_000401_e5430">
    <!-- ... -->
    <name type="author">Bouhier (Jean)</name>
    <trait>
        <p>président au Parlement de Dijon, membre de l'Académie
        ↪ française.</p>
    </trait>
    <!-- ... -->
</item>
```

Code source 4.7 – Le cas des métiers dont l'extraction est problématique

plus grand nombre de résultats. Enfin, ces deux exemples montrent qu'il n'est pas possible d'extraire et de traduire des informations sans prendre en compte ce qui sera pertinent pour le moteur de recherche de *Wikidata*. Il ne s'agit donc pas seulement d'extraire des informations, mais aussi de s'adapter avec ce moteur de recherche pour augmenter la probabilité d'obtenir un résultat valide.

4.1.5 Comment négocier avec le moteur de recherche de *Wikidata* ?

Comme cela commence à apparaître, l'extraction d'informations, lorsqu'elle vise à interagir avec des données externes, vient avec des difficultés supplémentaires. Il ne faut pas seulement extraire les informations ; leur extraction et structuration doivent permettre de lancer des recherches en plein texte, et donc de minimiser le bruit (les informations non pertinentes) et le silence (l'absence d'informations) de la part du moteur de recherche. Il faut donc traduire les informations extraites pour qu'elles correspondent au vocabulaire utilisé par *Wikidata*. Cette opération n'est pas anodine : si les catalogues de vente fonctionnent avec leurs propres catégories, le même peut être dit de *Wikidata* : certains types de données sont plus souvent référencées que d'autres et *Wikidata* utilise un vocabulaire qui lui est propre. Pour bien mener ce processus de traduction et de

structuration de l'information, il est nécessaire de bien connaître le fonctionnement de ce moteur de recherche pour mieux s'y adapter.

Comme cela a été dit, l'alignement avec *Wikidata* passe par l'utilisation de l'`API` mise en point par l'institution afin de lancer automatiquement des recherches en plein texte ; l'objectif est que le premier résultat retourné par le moteur de recherche soit le bon. La première chose à remarquer est que, contrairement à un moteur de recherche généraliste (comme *Google, QWant...*), ce moteur n'est pas compatible avec des requêtes approximatives. L'exemple 4.8 est pertinent à ce égard [4]. Dans de nombreuses entrées, comme c'est le cas ici, les fonctions d'une personne ayant participé à la révolution sont présentées de façon précise : Marc David Alba Lasource est décrit comme étant un « conventionnel girondin ». Cette mention, régulièrement présente dans les catalogues, pourrait être relevée en tant que telle. Cependant, lancer la recherche « Lasource conventionnel » ne retourne aucun résultat. Si la même recherche est lancée sur un moteur de recherche généraliste (ici, *QWant*), la page *Wikipedia* de Lasource fait partie des premiers résultats [5]. Cette différence dans les données retournées par les moteurs de recherche a deux explications : un moteur de recherche généraliste recherche les occurrences de mots, non seulement dans le titre de la page, mais aussi dans le corps du texte. Si le mot « conventionnel » est absent du titre, il est certainement à plusieurs reprises dans une notice biographique type *Wikipedia*. *Wikidata* ne contenant que des données, et pas de texte en tant que tel, l'indexation du corps du texte par le moteur de recherche interne à *Wikidata* n'est pas possible. Ensuite, la plupart des moteurs de recherche généralistes utilisent des méthodes de traitement du langage afin de simplifier la requête lancée par l'utilisateur.ice : les mots recherchés sont simplifiés, le moteur de recherche associe les termes recherchés avec d'autres termes « cooccurents », c'est-à-dire fréquemment utilisés ensemble [6]. Dans le cas du moteur de recherche de *Wikidata*, la requête de l'utilisateur.ice ne semble pas être retraitée : des signes de ponctuation ou des fautes de frappes influencent l'obtention d'un résultat, de même que l'usage de termes inadaptés.

Pour faire face à la « rigidité » relative du moteur de recherche de *Wikidata*, il est donc nécéssaire de préparer ses données au moment de leur extraction. En prenant le même exemple (4.8), un résultat correct peut être obtenu en remplaçant « conventionnel » par « politician » [7], pour rechercher sur *Wikidata* « Lasource politician ». Ici, la traduction de « conventionnel » en « politician » est d'autant plus intéressante que la date de naissance

4. Dans cet exemple, le prénom, « M.-D.-A. », n'est pas pris en compte pour se concentrer sur l'utilisation d'informations biographiques dans le `trait`.

5. Le 29/07/2022, c'est le troisième résultat ; le premier correspond à une vente aux enchères d'archives du conventionnel. Les moteurs de recherche pouvant être mis à jour régulièrement, il est possible que l'ordre des résultats change

6. **noauthor_moteur_2022**. Pour des analyses plus détaillées sur la construction d'ensemble de termes coocurrents via le développement de vecteurs de mots, voir **mikolov_efficient_2013** ; pour un article technique détaillant la classification et la sélection de résultats pertinents par apprentissage profond, voir **covington_deep_2016**

7. « Personnalité politique »

```xml
<item n="140" xml:id="CAT_000197_e140">
    <!-- ... -->
    <name type="author">Lasource (M.-D.-A.)</name>
    <trait>
        <p>célèbre conventionnel girondin, né près de Montpellier en
        ↪ 1762, guillotiné en 1793.</p>
    </trait>
    <!-- ... -->
</item>
```

Code source 4.8 – Le problème de l'approximation et de la traduction : Lasource, conventionnel

dans le catalogue (1762) ne correspond pas à celle indiquée sur *Wikidata* (1763). Dans un cas comme celui-ci, où certaines données sont incorrectes, il est important d'extraire un maximum d'informations pour que, si certaines requêtes ne rapportent pas de résultats, pouvoir en faire d'autres avec différents paramètres.

En conclusion, il faut retenir que le moteur de recherche de *Wikidata* n'admet pas d'erreurs, ni de requêtes partiellement erronées (dans l'exemple 4.8, où la date de naissance soit correcte, mais pas la date de décès) ; il ne prend pas non plus en compte la synonymie, ce qui veut dire qu'il n'améliore pas la requête lancée par un.e utilisateur.ice... Cela signifie que les termes utilisés dans une requête doivent être adaptés à ceux que *Wikidata* utilise. Les termes spécifiques utilisés dans les catalogues (« conventionnel »), mais aussi de nombreux titres militaires et de noblesse peu élevés (« capitaine », « marquis ») sont relativement rarement présents sur *Wikidata*. Lorsque les requêtes sont lancées, de tels termes sont donc abandonnés et parfois remplacés par des termes plus génériques : par exemple, « capitaine » est remplacé par « military », traduction anglaise de « militaire ». De même, des termes principalement en usage dans la langue française, comme « conventionnel » sont moins efficaces pour lancer des recherches.

4.1.6 Une approche prédictive

L'alignement avec *Wikidata* et l'extraction d'entités n'est donc pas une opération anodine : les données contenues dans les catalogues sont variées, autant par leur structure que par les informations qu'elles contiennent ; il peut être difficile à faire la traduction de données du XIX^{ème} s. en chaînes de caractères pouvant retourner des réponses valides sur *Wikidata* ; enfin, le l'alignement repose sur une bonne connaissance du moteur de recherche de *Wikidata*.

De plus, la technique utilisée dans l'extraction de données, reposant sur la détection de motifs à l'aide de expressions régulières et de tables de conversion, est une technique qui vient avec un certain nombre d'incertitudes. Avec ce genre de techniques, il est impossible

```
1        <name type="author">Verneuil (Charlotte Séguier duchesse
         ↪  de)</name>
```

Code source 4.9 – Peut-on identifier les différents éléments d'une phrase par détection de motifs ?

de « comprendre » ce qu'un élément signifie. Dans l'exemple 4.9, formellement, rien ne sépare le nom propre de la duchesse (« Séguier ») du nom de son duché (« Verneuil »). En s'appuyant sur une connaissance de la structure répétitive des entrées, il est uniquement possible de supposer que le nom entre parenthèses est un nom propre, tandis que le nom hors des parenthèses correspond au nom du duché. En bref, les méthodes de détection de motifs utilisées, peuvent uniquement inférer le sens d'un mot par rapport à sa position dans une phrase. Si cette technique implique une certaine incertitude, elle est cependant particulièrement adaptée à un corpus semi-structuré, comme c'est le cas des catalogues de vente de manuscrits, et à l'opération d'alignement avec *Wikidata*. Comme cela est expliqué plus bas, au fond, il n'est pas tellement important de distinguer le sens des différentes informations : ce qui a du sens, c'est que l'extraction et la structuration des informations permet de construire des chaînes de caractères à rechercher sur *Wikidata*. Identifier la fonction d'un mot n'est donc ici qu'un moyen – contrairement à de l'analyse lexicale, où la fonction des mots dans une phrase est signifiante –. En effet, repérer le rôle que tiennent les termes extraits (métier, prénom...) permet de mieux construire la chaîne de caractère recherchée sur *Wikidata*, en pouvant filtrer certaines informations (retirer les dates de vie et de mort, par exemple).

Étant donnée cette quantité d'incertitudes, l'approche suivie dans l'alignement avec *Wikidata* peut être qualifiée de « prédictive ». Par ce terme, il faut comprendre que il n'y a pas, de certitude totale dans le processus d'extraction et de traduction des données. Il n'est pas possible de récupérer avec une certitude totale le bon identifiant. L'objectif cet algorithme n'est donc pas de trouver la « bonne » réponse. Il est de construire une chaîne de caractère dont on prédit qu'elle apportera un résultat pertinent. De la même manière, la phase de préparation des données est un processus qui sélectionne et normalise certaines informations dont on considère – après un long processus de test et d'essais – qu'elles seront pertinentes dans l'obtention des bons résultats. Enfin, le premier rôle des tests est de quantifier les prédictions. Ils répondent à la question : étant donné les résultats obtenus lors des tests, quelle est la probabilité que la prochaine chaîne de caractères recherchée retourne un résultat pertinent ? Cette approche prédictive implique nécessairement un degré d'incertitude, et donc le développement d'algorithmes flexibles qui cherchent à minimiser le bruit.

Être conscient de la nature prédictive de ce processus et quantifier la qualité des algorithmes à l'aide de tests permet cependant de prendre de meilleures décisions tech-

niques. La lecture distante et la détection de motifs supposent d'avancer « à l'aveugle », en s'appuyant sur sa connaissance de la structure du texte pour extraire les bonnes informations. Étant donné qu'il est impossible d'être totalement certain que les bonnes données ont été extraites, l'étape suivante – le lancement des requêtes sur l'`API` – doit malléable et s'adapter aux données disponibles. C'est pourquoi le parti pris a été de concevoir un algorithme qui continue de lancer des requêtes en retirant des paramètres tant qu'un identifiant n'a pas été trouvé.

4.2 Un algorithme de détection de motifs pour préparer et structurer les données

Avant de chercher à récupérer un identifiant *Wikidata* via l'`API`, un algorithme se charge de traduire et de structurer les données : à partir d'un nom et de son éventuelle description, un dictionnaire qui contient les informations de manière structurée est construit. Cette étape était initialement censée être une simple extraction d'information : à partir du `name` et du `trait`, un ensemble d'informations étaient mises bout à bout afin de former une chaîne de caractères à rechercher sur l'`API`. Le processus s'est complexifié pour intégrer l'extraction, la traduction et la structuration des données. En construisant un dictionnaire à partir de texte, il est possible de savoir précisément quelles données sont disponibles pour lancer des requêtes ; plusieurs requêtes peuvent alors être lancées sur l'`API` avec différents paramètres, ce qui permet d'augmenter les probabilités d'obtenir un identifiant valide.

4.2.1 Présentation générale

Les formats d'entrée et de sortie

Le but de l'extraction de données permet de transformer la représentation `TEI` visible en 4.10 – représentée sous forme d'un `TSV` pour faciliter la lecture des données – au dictionnaire visible en 4.11. Voici la significtion des différentes clés[8] du format de sortie :

— `fname` : cette clé permet d'accéder au prénom d'une personne. Les données contenues dans cette clé viennent du `name`. `fname` est l'abréviation de « first name ».

— `lanme` : cette clé permet d'accéder au nom de famille de quelqu'un. C'est cette information, extraite du `name`, qui est centrale aux requêtes. `lname` abréviation de « last name »)

8. Une clé de dictionnaire est l'élément à gauche des « : » ; la clé permet d'accéder à la valeur, visible à droite du « : », ce qui permet d'associer des valeurs entre elles, et donc de stocker des objets ou de remplacer une clé présente dans un texte par une valeur, par exemple.

```xml
<item n="271" xml:id="CAT_000327_e271">
    <!-- ... -->
    <name type="author">Turenne (Henri de La Tour d'Auvergne vicomte
    ↪ de)</name>
    <trait>
        <p>illustre maréchal de France, né en 1611, tué en 1675.</p>
    </trait>
    <!-- ... -->
</item>
```

Code source 4.10 – L'entrée `XML-TEI` à partir de laquelle des données sont extraites

— `nobname_sts` : cette clé contient un nom de famille noble. Dans ces cas, un titre de noblesse est présent dans les entrées de catalogue (seuls les titres de noblesse les plus importants sont extraits du dictionnaire, ce qui n'est pas le cas ici). Les informations contenues ici proviennent du `name`. Cette clé est l'abréviation de « nobility name_status »

— `status` : le statut d'une personne, soit son titre de noblesse (ici, le titre « vicomte » n'a pas été extrait car il est rarement présent sur *Wikidata*). Les informations contenues ici proviennent en général du `name` et parfois du `trait`.

— `dates` : les dates de naissance ou de mort d'une personne (seules ces dates sont conservées). Ces informations proviennent du `trait`.

— `function` : la fonction d'une personne, soit, en général, son métier ou son occupation principale. Cette information provient du `trait`.

— `rebuilt` : un booléen indiquant si un prénom a été reconstruit à partir d'initiales ou non.

Comme cela a été dit auparavant, le nom attribué à ces clés n'est pas systématiquement indicateur des valeurs qui y sont associées : si l'entrée de catalogue correspond à une personne, alors les clés correspondent aux informations qu'elles contiennent. Si l'entrée de catalogue ne correspond pas à une personne, ces clés seront également utilisées. Ce qui est important, c'est la hiérarchie d'importance entre les différentes clés : `lname` est la clé centrale et contient presque toujours des informations, `fname` des données secondaires et `date` des dates. Les autres clés sont rarement utilisées si l'entrée de catalogue ne correspond pas à une personne.

Présentation de l'algorithme d'extraction d'informations

L'algorithme détaillé ci dessous est présenté sous forme graphique dans la figure 4.3. Cette étape peut être séparée en deux parties différentes : l'extraction d'informations nominatives du `name` et la récupération de données biographiques du `trait`.

```
{
    "fname": "henri ",
    "lname": "la tour d'auvergne",
    "nobname_sts": "Turenne ",
    "status": "",
    "dates": "1611 1675 ",
    "function": "marshal",
    "rebuilt": False
}
```

Code source 4.11 – La sortie `JSON` correspondante

L'extraction d'informations du `name` est l'étape plus complexe. La difficulté tient au fait que cette balise peut contenir des informations variées et structurées de façon très différente. Cela demande d'identifier des motifs récurrents et de les repérer dans le texte à différents degrés et à différentes étapes. Dans un premier temps, les prénoms sont systématiquement extraits. Ils sont toujours détectés – même si la balise ne contient le nom d'une personne : cette extraction permet justement d'identifier le type de données contenues dans le `name`. Les prénoms sont repérés à l'aide de plusieurs expressions régulières qui permettent d'identifier des prénoms complets et abrégés, qu'ils soient composés ou non. Cette détection prend en compte les différents types d'abréviations possibles (un prénom composé peut être entièrement abrégé ; à l'inverse, seulement un des prénoms peut être abrégé) et les différentes typographies (séparer les prénoms avec des traits d'union ou non, par exemple). Dans le cas où un prénom serait abrégé, il est si possible reconstruit : des initiales sont remplacées par un nom complet ; ce processus est présenté plus en détail ci-dessous. Cela permet d'augmenter le taux de réussite dans l'alignement avec des identifiants *Wikidata*, mais vient avec plusieurs difficultés techniques, comme nous le verrons. Une fois ce nom extrait, le type d'information contenue dans le `name` doit être identifié : en fonction du type d'information (géographique, historique, nominative...), différents traitements sont mis en place. Cette identification se fait par détection de motifs augmentée par l'usage de tables de conversion [9] et de listes contenant du vocabulaire spécifique. Listes et tables étant classées thématiquement, il est possible, par un processus éliminatoire, d'identifier avec certitude le type d'information contenue dans le `name`. Le traitement du `name` dépend grandement de cette détection : si cette balise contient des éléments géographiques ou historiques, l'extraction d'informations repose en grande partie sur les tables de conversion. S'il s'agit en revanche d'un nom de personne, il est alors nécessaire d'identifier les différents types de données nominatives (prénom, nom de famille, nom de famille noble...) pour bien structurer les données. En effet, c'est de cette structure que dépend la bonne construction du dictionnaire, et donc la constitution de

9. Pour un exemple de table de conversion, voir D.1

requêtes adaptées à l'`API`. Ce processus d'extraction des données s'appuie majoritairement sur une détection de motifs. Le motif déterminant est la présence ou non dans parenthèses dans le nom. Comme nous le verrons, si un nom contient des parenthèses, les informations sont bien plus structurées que s'il n'en contient pas. Les informations peuvent alors être extraites avec une bien plus grande granularité.

Une fois les informations nominatives extraites du `name`, il reste à extraire les données biographiques pertinentes du `trait`. Cette étape, plus simple que la précédente, vaut principalement pour les entrées où c'est l'auteur.ice d'un document qui est mentionné.e dans le `name`. Les seules informations extraites concernent les dates de vie et de mort des personnes, ainsi que son métier. Quelques difficultés techniques subsistent cependant : il faut notamment distinguer une date de naissance/décès d'une autre date, afin de diminuer le bruit ; de plus, il faut réussir à extraire de façon automatique l'occupation principale d'une personne, lorsque plusieurs personnes sont mentionnées (« militaire » et « auteur », par exemple). La résolution de ces deux problèmes repose sur une bonne connaissance du corpus de catalogues et de la structure des `trait`.

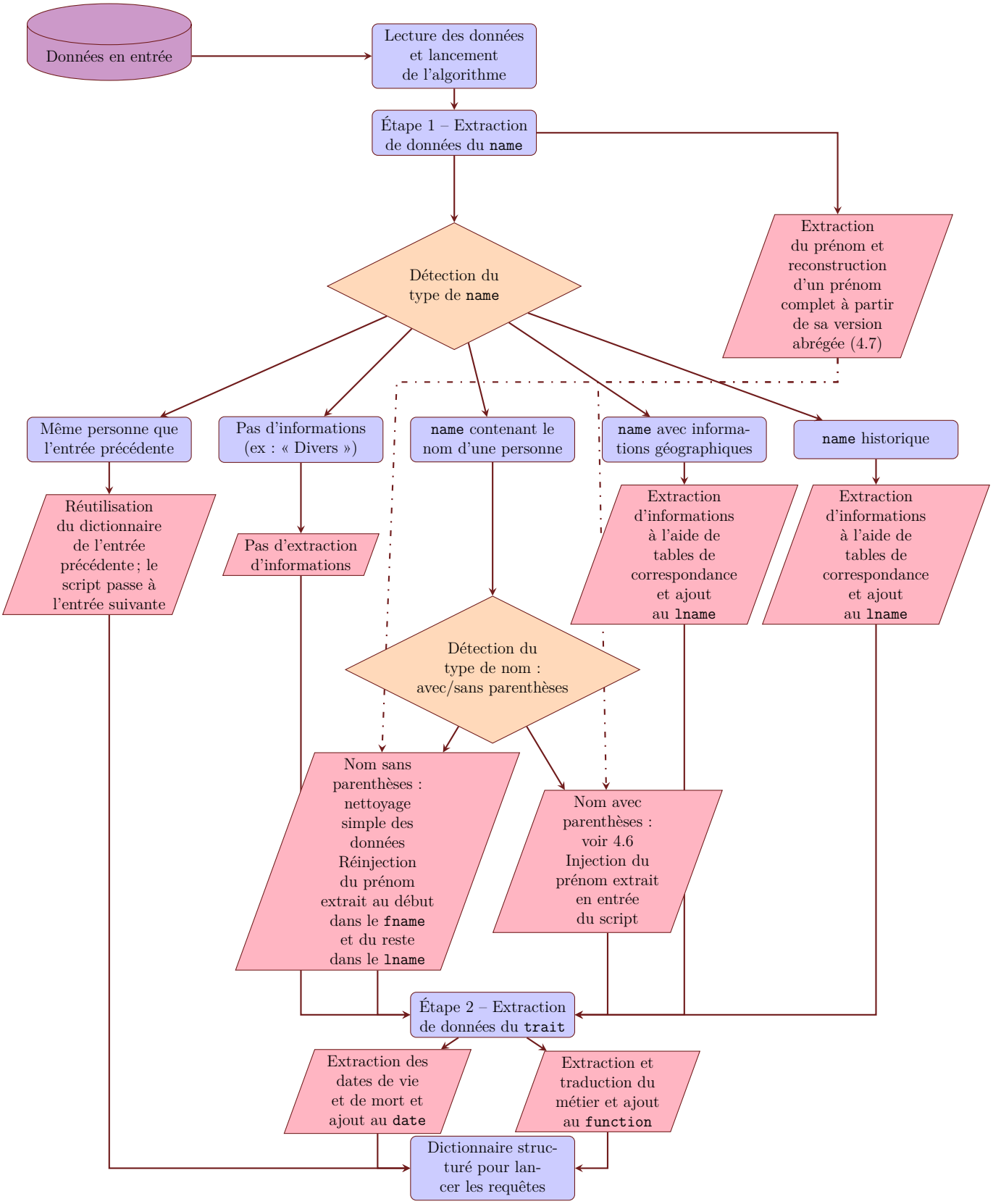


FIGURE 4.3 – Processus d'extraction d'informations du `name` et `item`

4.2.2 Identifier le type de nom

Les éléments `tei:name` contiennent le titre donné à l'item vendu. Si c'est souvent le nom de l'auteur.ice du document, ce n'est pas toujours le cas. L'extraction d'éléments du `name` dépend, comme cela a été dit, de l'identification du « type » de nom. La décision a été prise de classer tous les `name` en cinq catégories.

Les noms génériques : ces éléments ne contiennent pas d'informations précises. Les entités *Wikidata* étant spécifiques plutôt que génériques, il n'est pas certain que les entrées puissent être alignées avec *Wikidata* ; si des entités *Wikidata* correspondent à ces éléments, les informations qu'elles contiennent seront probablement trop génériques pour être utilisables dans un contexte économétrique. Lorsque cela est possible, l'alignement est quand même fait ; c'est le cas par exemple pour les `name` contenant la mention de chartes. Dans ces cas, un dictionnaire vide est retourné et l'alignement avec *Wikidata* n'aura pas lieu. Une exception est faite dans le cas chartes, où un alignement est fait avec l'entité *Wikidata* génériques. Dans cette catégorie se trouve par exemple :

— `<name type="author">DIVERS</name>`

Les noms de type « Le même » ou « La même » ; lorsqu'il y a cette valeur dans le `name`, l'auteur.ice est la même personne que l'auteur.ice de l'entrée de catalogue précédente. Dans ce cas, le dictionnaire de cette entrée est réutilisé. Par exemple :

1. `<name type="author">Le même</name>`.

Les noms géographiques ; ces entrées sont détectées à l'aide de tables de conversion et de listes. Celles-ci sont classées thématiquement : liste d'anciennes colonies françaises (D.4), de départements français du XIX$^{\text{ème}}$ s.(D.2), d'anciennes provinces françaises (D.5) et de pays (D.3). Dans ce cas, un alignement avec une entité *Wikidata* est possible ; cependant, il n'est pas toujours envisageable de retrouver l'entité précise. En effet, un `name` peut contenir une mention d'une donnée géographique, mais également d'autres détails. C'est par exemple le cas dans le quatrième exemple ci-dessous. Il faut alors aligner le `name` avec son équivalent générique sur *Wikidata* (l'exemple ci-dessous, par exemple, a été aligné uniquement avec l'entité « Paris »). Dans cette catégorie se trouvent :

— `<name type="other">AISNE (département de 1')</name>`

— `<name type="author">Bourbonnais. </name>`

— `<name type="author">Paris : Musée royal du Louvre</name>`

— `<name type="author">Garde nationale parisienne en 1792 (brevet de volontaire de la)</name>`

Les noms correspondant à des évènements historiques. Là encore, une table de conversion est utilisée (D.6). Ici, une difficulté apparaît cependant : du fait de la varité des évènements historiques mentionnés dans les entrées de catalogue, il n'est pas possible

d'enregistrer l'ensemble des évènements dans des tables afin de permettre une détection de tous les évènements. Les `name` ont donc été analysés pour extraire les évènements les plus importants. Ensuite, comme pour les termes géographiques, il n'est pas possible de donner aux tables de conversion une granularité suffisamment fine pour contenir toutes les données possibles. Des alignements partiels ont donc été faits : le premier exemple ci-dessous a été aligné avec l'entité *Wikidata* « Révolution française ».

— `<name xmlns="http://www.tei-c.org/ns/1.0" type="other">THÉATRE RÉVOLUTIONNAIRE</name>`

— `<name type="author">COMMUNE DE 1871.</name>`

— `<name type="other">Siège de La Rochelle en 1628</name>`

Les noms de personnes. Ceux-ci ne sont pas simples à traiter : ils peuvent contenir de nombreuses informations : deux noms de famille (usuels et nobles), titres de noblesse, plusieurs prénoms. Ils ont également une structure variée, comme cela apparaît dans les exemples ci-dessous : les noms peuvent être écrits en utilisant des parenthèses ou non ; un prénom peut être écrit en entier, comme dans premier exemple, entièrement abrégé (comme dans l'exemple 3) ou encore partiellement abrégé, ce qui est le cas dans le troisième exemple. Ces différences, qui ne posent pas de problème à un regard humain, sont autant de problèmes techniques. En effet, la détection de motifs fonctionne uniquement sur des critères formels, ou structurels. Il faut donc, en s'appuyant sur la structure des documents, réussir à distinguer un prénom d'un nom de famille, un nom de famille d'un autre, ou encore un nom abrégé d'un nom complet.

— `<name type="author">Humboldt (le baron Alexandre de)</name>`

— `<name type="author">Taccani Tasca (madame la comtesse)</name>`

— `<name type="author">LEGOUVÉ (G. M. J. B.)</name>`

— `<name type="author">LOUIS XVIII</name>`

— `<name type="author">Duras (Emm.-F. de Durfort, duc de)</name>`

Si l'on pose l'extraction d'informations nominatives sur une personne comme étant l'objectif principal, une difficulté apparaît vite : qu'est-ce qui distingue un nom de personne d'un des autres types de noms ? Les éléments `name` sont voués à contenir des noms propres ; chercher à distinguer les noms propres des noms communs n'a donc pas d'intérêt. Cela est d'autant plus que la graphie varie d'un catalogue à l'autre : dans certains, les majuscules sont signifiantes et pourraient permettre de distinguer noms communs et noms propres, tandis que dans d'autres, l'intégralité du `name` est en majuscule. Une détection de motifs à partir de simples critères formels (du type : « Un nom de personne est un ou plusieurs mots commençant par des majuscules ») n'est pas opérante pour ce corpus. Il n'est pas non plus possible de définir un nom positivement, puisqu'il n'existe aucun critère définitoire

pour un nom propre. Enfin, à cette étape, il n'est pas non plus possible de s'appuyer sur l'extraction de prénoms. L'extraction de prénoms se base, comme on le verra, sur de la détection de motifs ; là encore, ce qui est identifié comme un prénom peut tout aussi bien être un nom propre, et il n'existe à ce stade aucune possibilité pour distinguer un nom propre d'un autre. C'est à ce stade qu'a été décidée l'utilisation de tables de conversion et de listes contenant du vocabulaire thématique : en identifiants des références récurrentes à des évènements ou des lieux dans les `name`, il devient possible de définir les noms de personne négativement. En définitive, un nom de personne, c'est donc ce qui n'est pas autre chose et la détection du type de nom fonctionne donc de manière éliminatoire.

L'algorithme mène donc une série de tests, cherchant à détecter si un `name` rentre dans telle ou telle catégorie. Du fait du fonctionnement technique de `python`, une série de tests éliminatoires doit aller du cas le plus particulier au cas le plus générique afin d'éviter les faux positifs : une fois qu'un élément a été détecté comme appartenant à une catégorie, il ne peut plus être réassigné à une autre. C'est pourquoi l'algorithme commence par chercher à classer les éléments dans les catégories où le taux d'erreur est le plus faible, pour ensuite finir par la catégorie la plus générique : celle des noms de personne. : c'est dans ces catégories que le taux d'erreur est le plus faible.

Le script commence donc par chercher à classer un `name` dans les catégories où les informations sont écrites plus ou moins toujours de la même manière. Il cherche d'abord à identifier si le `name` correspond à « Le même » ou « La même ». Dans ce cas, le `name` est le même que celui de l'entrée précédente et c'est ce dictionnaire qui est réutilisé. Ensuite, les entrées génériques sont détectées. Comme elles contiennent toujours des informations écrites de la même manière (« Documents divers »), le taux d'erreur est là encore très faible. Ces entrées génériques ne contiennent pas d'informations spécifiques ; si un `name` appartient à cette catégorie, alors l'algorithme n'extrait pas d'informations.

Ensuite, si un `name` n'appartient ni à l'une ni à l'autre catégorie, alors l'algorithme cherche à classifier un nom en différentes catégories à l'aide de tables de comparaison et de listes contenant du vocabulaire spécifiques. Ces tables et listes sont classées en deux catégories (données historiques et géographiques) afin de définir des traitements spécifiques ; l'algorithme cherche d'abord à identifier des entrées géographiques, puis historiques. En effet, un bien plus grand nombre de tables contenant des données géographiques existe [10], ce qui augmente les possibilités d'un classement correct. Enfin, le script cherche à identifier des informations historiques dans un `name` (D.6). Si une donnée géographique ou historique est repérée, alors équivalents normalisés de cette donnée sont ajoutés au dictionnaire grâce à l'usage de tables de conversions.

Par processus d'élimination, si aucune de ces informations n'a été détectée, alors il n'est plus possible de classer le `name` dans aucune autre catégorie. Il est alors considéré que

10. Nom d'anciennes provinces françaises (D.5), de départements du XIX$^{\text{ème}}$ s. (D.2), d'anciennes colonies (D.4) et de pays (D.3)

le contenu du `name` est le nom d'une personne. L'extraction d'informations se fait ici plus complexe [11], comme nous le verrons : le script traite le nom différemment en fonction de sa structure (présence ou non de parenthèses dans le nom), puis extrait d'éventuels titres de noblesse, noms de famille noble et nom de famille usuel. Enfin, il extrait des prénoms et cherche à reconstruire un prénom complet à partir de son abréviation. Le processus étant éliminatoire, il n'y a bien sûr aucune certitude totale que le contenu du `name` soit bel et bien le nom d'une personne ; il n'est cependant plus possible de mieux catégoriser les éléments. Cela n'est pas non plus extrêmement important, puisque cet algorithme de classification a un rôle fonctionnel et n'impacte pas la manière dont les identifiants sont récupérés depuis l'`API` *Wikidata*. Il permet principalement d'adopter un fonctionnement modulaire en cherchant à détecter des motifs spécifiques dans les `name` en fonction de la catégorie à laquelle ils appartiennent. La détection de motifs étant « aveugle », le traitement qui est fait pour cette catégorie peut être mis à profit de différents types de données : les mêmes motifs peuvent être recherchés et extraits de différents types d'entrées.

À l'issue de cette phase de classification, il est possible de mieux connaître les types de `name` et leur répartition dans le corpus. Les noms de personnes sont très majoritaires dans les `name` : ils représentent 80634 entrées sur un total de 82913, soit 97,25% du corpus. C'est donc pour ce type de données que l'extraction des données a été pensée, ce qui permet d'extraire des informations avec une granularité plus fine qu'avec le reste du corpus. Viennent ensuite les noms de lieux (1406 entrées) et les éléments divers (550). Pour finir se trouvent les évènements historiques (232 entrées) et les éléments vides, soit 92 entrées [12]. Bien que, comme cela a été dit, la classification des `name` en différents types soit purement fonctionnelle, ces chiffres permettent d'avoir une meilleure connaissance du corpus, et mieux comprendre à partir de quel type de données se fait l'alignement *Wikidata*. Cette classification montre également d'intérêt de la détection de motifs augmentée de tables de correspondances : bien que ce soit une méthode « légère », elle peut efficacement, sur un corpus semi-structuré, être à la base d'une extraction précise de données.

4.2.3 Le traitement des noms de personnes

Comme cela a déjà été dit, le processus d'extraction d'informations du `name` est relativement simple pour les entrées qui ne contiennent pas de noms de personnes : il s'agit principalement de remplacer des informations non-normalisées par leur équivalent normalisé, à l'aide d'un dictionnaire. Pour les noms de personne cependant, le processus est plus complexe : il demande d'identifier titres de noblesse et différents types de noms ; de plus, de nombreux prénoms sont abrégés pour gagner de la place dans les catalogues. Pour

11. Pour une représentation graphique de cette étape, voir 4.6

12. Ces chiffres proviennent d'un calcul réalisé à partir du script développé pour identifier le type d'entrée. Ils ont été produits afin de réaliser une représentation graphique du type d'entrée, visible en annexes (C.2)

augmenter le taux de réussite de l'alignement avec *Wikidata*, une méthode a été développée pour reconstruire un nom complet à partir de son abréviation. Après avoir présenté les différentes étapes de l'extraction d'informations nominatives, cette partie détaillera ce processus de reconstruction du prénom. Dans ces deux étapes, la nature semi-structurée des documents est centrale, puisqu'elle permet d'identifier avec une grande précision les différentes informations.

Trouver des solutions adaptées à différents types de noms

Pour extraire des informations d'un `name` et dans leur un statut ou d'une signification spécifique, il est nécessaire d'identifier ce qui distingue un prénom d'un nom de famille ou d'un titre de noblesse. Une première difficulté apparaît vite : formellement, il n'y a pas nécessairement de différence entre ces informations. Dans l'exemple 4.10, le `name` correspond à :

```
1 <name type="author">Turenne (Henri de la Tour d'Auvergne vicomte
  ↪ de)</name>
```

Code source 4.12 – Le `name` de l'exemple 4.10

Les informations à extraire de cet élément sont visibles dans la figure 4.4. Si chaque mot est pris individuellement, il est difficile de les distinguer l'un de l'autre. Le nom de famille noble, le prénom et le nom de famille usuels débutent tous par une majuscule. Il est ici impossible de trouver un motif distinctif pour un nom de famille ou pour un prénom. Cela est d'autant plus vrai qu'un nom peut être composé ; ici, le nom de famille usuel est composé de deux mots débutant par une majuscule et séparés par un « d' ». Étant donné la taille du corpus, il n'est pas non plus possible d'utiliser des tables de conversions pour identifier ce qui est un nom ou un prénom, comme cela a été fait pour les lieux et les évènements géographiques. La seule information qui peut être traitée à l'aide d'une table de conversion est le titre de noblesse, puisque ces titres existent en nombre limité. Par conséquent, aucun élément à l'intérieur des termes à relever ne permet de les identifier. Là où ils peuvent cependant être distingués, c'est au niveau de leur place relative au sein de la phrase. Pour reprendre l'exemple 4.4, le nom de famille noble se retrouve hors de la parenthèse, au tout début de la phrase ; les autres informations sont dans la parenthèse, où le prénom est suivi du nom de famille et du titre. S'il n'est pas possible d'identifier la valeur d'un nom à partir de ses caractéristiques propres, il est donc possible de la déduire de sa position dans la phrase. C'est ici qu'une bonne compréhension de la structure des `name` devient essentielle.

Il est donc nécessaire d'identifier les différentes structures possibles pour un `name` contenant un nom de personne. Celles-ci sont visibles en 4.5. Pour les besoins de l'algo-

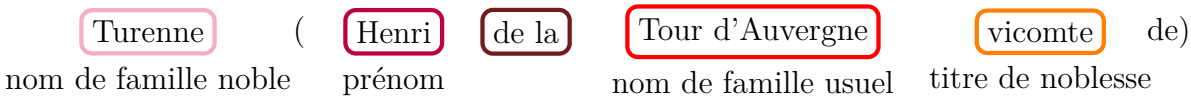


FIGURE 4.4 – Les différentes parties du `name`

rithme, deux structures principales ont été retenues : les `name` contenant des parenthèses et eux n'en contenant pas [13]. Comme nous le verrons, d'autres subdivisions existent ensuite.

Les `name` contenant des parenthèses ont une structure bien plus claire que ceux n'en contenant pas ; il est alors tout à fait possible d'identifier le rôle joué par les différents éléments de cette balise. L'organisation entre les différents types de noms est la même dans les exemples 4.13 et 4.14, qui contiennent tous les deux des parenthèses. Le nom le plus important, sous lequel une personne est connue au XIX$^{\text{ème}}$ s., est contenu entre parenthèses et au début de l'entrée, parfois en majuscules. Le ou les noms suivants sont, eux, à l'intérieur des parenthèses. Le premier nom au sein des parenthèses est toujours le prénom ; ensuite viennent des informations complémentaires. Ces deux exemples, cependant, forment deux sous-catégories dans le groupe des noms sans parenthèses :

— Le premier exemple (4.13) contient uniquement des noms de personnes nobles. Dans ce cas, le nom hors parenthèses correspond au nom de famille noble ; le nom de famille usuel d'une personne est contenu à l'intérieur des parenthèses. Ce détail est important, parce que les personnes sont souvent référencées sur *Wikidata* par le nom de famille usuel. Lancer une recherche sur l'API avec les noms de famille usuel et noble à la fois ne retournera pas toujours de réponse, alors que ne rechercher qu'un des deux noms peut permettre d'obtenir le bon résultat. C'est pourquoi, dans ces cas, les deux noms sont distingués : le nom de famille usuel est associé avec la clé `lname` du dictionnaire, et le nom noble lié au `nobname_sts`.

— Les personnes dans le second exemple (4.14) ne sont pas nobles, ce qui simplifie beaucoup l'extraction d'information. Il suffit de stocker la partie entre parenthèses hors du `lname` ; le prénom entre parenthèses, identifié grâce à l'algorithme d'extraction des prénoms, est stocké dans le `fname`. Parfois, d'autres informations sont contenues dans les parenthèses ; si elles sont signifiantes (comme dans le cas d'Alexandre Dumas père, où il est important de faire la distinction d'avec son fils), elles sont extraites. Sinon, ces informations supplémentaires sont abandonnées.

Les `name` ne contenant pas de parenthèses (exemple 4.15) ne peuvent être traités avec la même granularité, puisqu'ils ne présentent pas de caractères récurrents pour faire la différence entre leurs différentes parties. Il est donc impossible de s'appuyer de façon récurrente sur la position des différents noms dans la phrase pour déduire leur signification. Cependant, il faut également remarquer que les éléments ne contenant pas de parenthèses contiennent en général bien moins d'informations que ceux qui en contiennent. Chercher

13. Comme cela étant déjà visible dans la figure 4.3

```
<name type="author">Turenne (Henri de la Tour d'Auvergne vicomte
↪ de)</name>
<name type="author">HUMBOLDT (Alexandre baron de)</name>
<name type="author">Tascher (Pierre-Jean-Alexandre-Jacquemin comte
↪ Imbert de)<name>
```

Code source 4.13 – Trois exemples de `name` avec titres de noblesse

```
<name type=author>Viardot (Pauline)</name>
<name type=author>Verdi (Giuseppe)</name>
<name type=author>Sobieski (Thérèse-Cunégonde)</name>
<name type="author">CAUCHY (le bon Alex.)</name>
<name type="author">DUMAS (Alex. père)</name>
```

Code source 4.14 – Cinq exemples de `name` sans titres de noblesse

à tout prix à identifier une structure à ces entrées afin de pouvoir avoir un dictionnaire complet à partir duquel lancer plusieurs requêtes n'a pas non plus nécessairement d'intérêt.

```
<name type="author">Sophie</name>
<name type="author">Henri VI</name>
<name type="author">DUPUIS</name>
```

Code source 4.15 – Trois exemples de `name` sans parenthèses

Identifier et extraire les informations nominatives

C'est sur les différents critères présentés dans la partie précédente et résumés sous forme graphique dans la figure 4.5, que se base l'extraction d'informations nominatives du `name`.

Dans un premier temps, l'algorithme cherche à extraire des informations d'un `name` contenant des parenthèses (étape représentée dans le graphique : 4.6). Il y a alors quatre données à identifier et à extraire : le nom de famille noble, le nom de famille usuel, le prénom et le titre de noblesse. L'algorithme commence par extraire des informations du texte contenu entre parenthèses avant de passer au texte hors parenthèses : il identifie un titre de noblesse avant d'extraire un prénom et éventuellement un nom de famille ; pour finir, le nom hors parenthèses est extrait.

L'extraction du titre de noblesse est l'étape la plus simple, puisqu'elle repose sur une table de conversion (D.7) : l'algorithme cherche à identifier un titre de noblesse parmi les clés de ce dictionnaire. Si des titres sont trouvés, alors leur version normalisée figurant en valeur de ce dictionnaire est ajoutée aux données extraites. Il est à noter que, comme tous

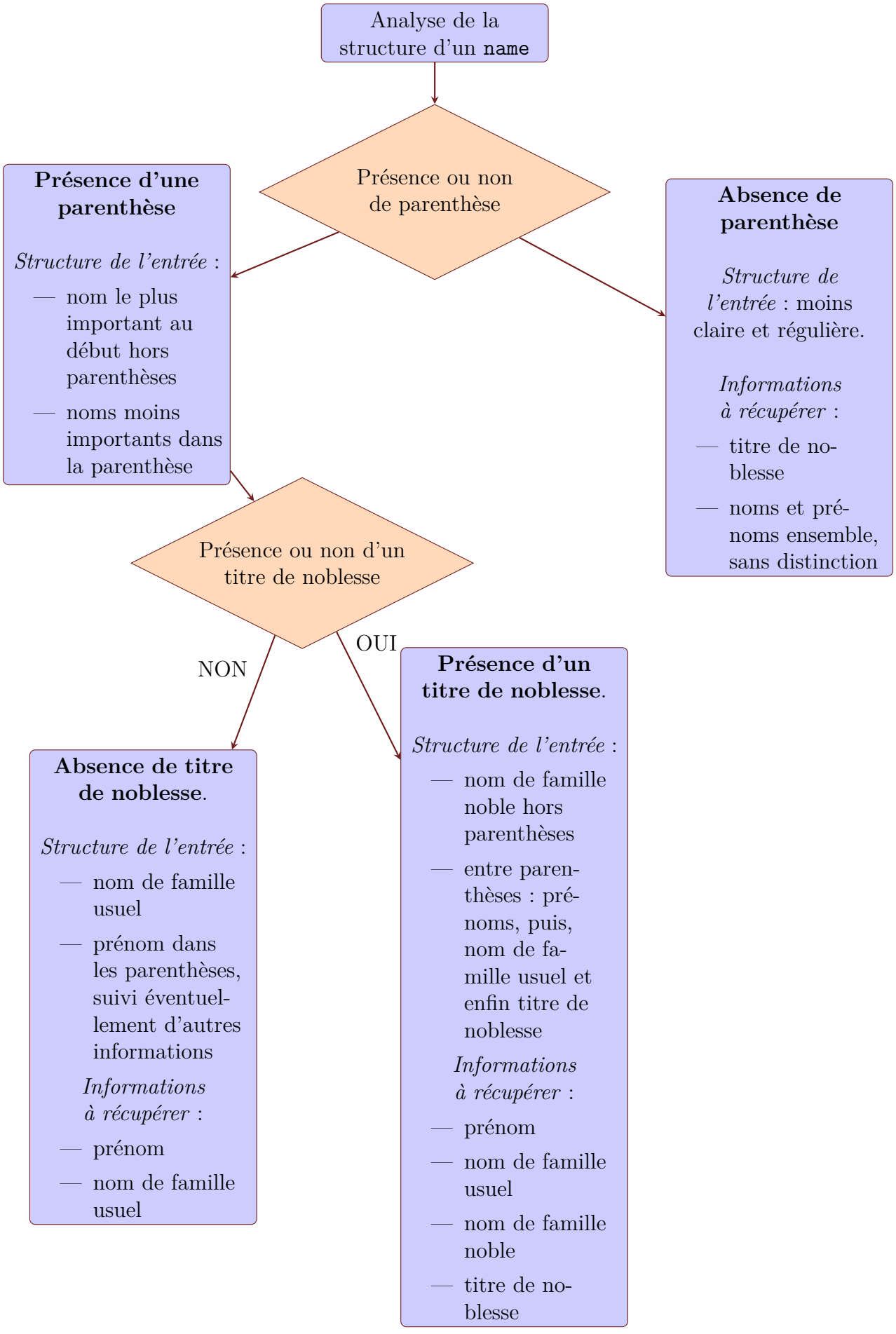


FIGURE 4.5 – Différentes structures possibles pour un `name`

les titres de noblesse les plus importants permettent d'obtenir des résultats sur *Wikidata*, aucune valeur n'est associée aux titres les moins importants ; dans ce cas, aucun titre n'est extrait, puisqu'il risque d'empêcher l'obtention d'un résultat. Cependant, même des titres de noblesse n'ayant pas d'équivalent, et qui ne seront donc pas extraits, figurent sur cette table de conversion : la table permet d'identifier la présence d'un titre de noblesse, ce qui est nécessaire pour extraire les autres informations.

Si un titre de noblesse a été trouvé, un processus éliminatoire est engagé afin d'identifier le prénom et le nom de famille usuels contenus entre parenthèses. Comme cela apparaît dans les exemples plus haut, le texte entre parenthèses peut contenir du bruit : des mots comme « du, le... » qui n'aident pas à obtenir de résultats sur *Wikidata*, ou encore des noms communs et des attributs des personnes. Il faut détecter tous les noms propres, situer leur rôle et les extraire. Il peut être tentant d'extraire seulement les noms propres à l'aide d'expressions régulières – c'est à dire, d'identifier tous les mots commençant par une majuscule – et d'abandonner le reste. Cependant, certains noms communs peuvent être écrits avec une capitale ; en suivant cette méthode, il y aurait donc un risque d'inclure du bruit dans les données utilisées pour lancer les requêtes. L'algorithme lance donc une série de simplifications du contenu entre parenthèses : il commence d'abord par supprimer les données déjà extraites : le prénom ou son abréviation identifiées et le titre de noblesse ; ensuite, il supprime l'ensemble des termes « auxilliaires » (« le, la, dit... »), qu'ils aient des capitales ou non ; enfin, tous les noms communs restants sont supprimés. Il ne reste alors du texte entre parenthèses qu'un ou plusieurs noms commençant par une majuscule et considérés comme des noms propres ; quelle valeur donner à ce texte ? Étant donné que le prénom a déjà été identifié, de même que le titre de noblesse, le texte entre parenthèses ne peut être qu'un nom de famille ; on considère alors que le texte entre parenthèses est un de nom de famille usuel ; le texte hors parenthèses est alors le nom de famille noble.

Si un `name` contient des parenthèses mais pas de titre de noblesse, le processus est analogue à celui décrit ci-dessus – mais plus simple, puisque le texte entre parenthèses a tendance à être bien plus complexe pour les personnes nobles. Le bruit est successivement détecté et supprimé à l'aide d'expressions régulières ; le prénom déjà extrait est supprimé. À ce stade, une deuxième extraction de prénoms a lieu : les prénoms étant parfois complexes et écrits d'une manière variable (avec ou sans parenthèses...), il arrive que ceux-ci ne soient pas entièrement extraits. Cette seconde extraction pose une légère difficulté, puisque l'on dispose alors de deux noms propres ayant la valeur d'un prénom. Il faut déterminer la relation entre ces deux « prénoms » : les mettre simplement bout-à-bout risquerait de créer des noms et prénoms dans le désordre ; dans ces cas, le moteur de recherche de *Wikidata* ne retournerait pas de résultat. Par conséquent, c'est en fonction de la position relative des deux « prénoms » que le prénom final est reconstruit : celui qui se situe au début du texte entre parenthèse est positionné en premier dans le prénom reconstruit. Pour finir, c'est le texte hors parenthèses qui est extrait, soit le nom de fa-

mille. En l'absence de titre de noblesse, il n'est pas nécessaire de faire la différence entre différents noms de famille ; le nom hors parenthèses ne peut être qu'un nom de famille usuel.

Si un nom ne contient pas de parenthèses, le processus d'extraction est à la fois plus simple et moins qualitatif : il est tout simplement impossible de classer les différentes données extraites par détection de motifs. Dans ce cas là, le bruit supprimé du `name` ; l'objectif est de pouvoir lancer des recherches sur *Wikidata* avec les données les plus propres possibles, afin de maximiser les possibilités d'obtenir un résultat correct.

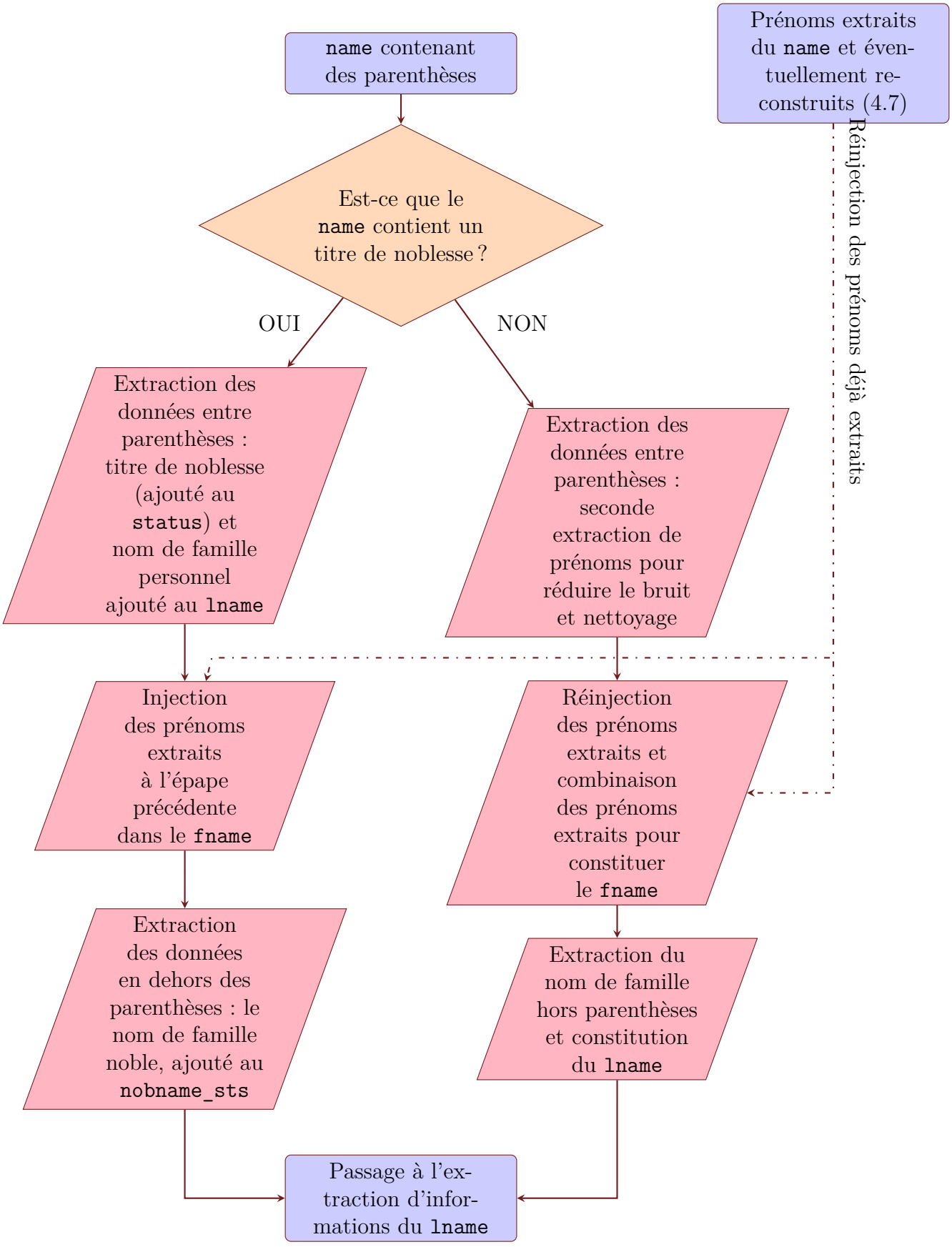


FIGURE 4.6 – Extraction d'informations d'un `name` contenant des parenthèses

Reconstruire un prénom complet à partir de son abréviation

Cette phase essentielle vise à corriger un problème dans les données : le nom d'une personne est souvent abrégé. Cela pose un problème pour l'utilisation ultérieure de l'`API` : le prénom est une information essentielle pour identifier l'alignement avec la bonne entité sur *Wikidata*. Sans ce prénom, le risque est grand que le premier résultat obtenu à l'aide du moteur de recherche de *Wikidata* ne soit pas celui de la personne décrite dans le `name`, mais celui d'un.e autre membre de la même famille. De plus, si une requête est lancée avec un prénom abrégé (les initiales d'une personne, par exemple), elle ne retourne pas de résultat [14]. Le choix a donc été fait, pour chaque entrée, d'essayer de reconstruire un prénom complet à partir de son abréviation. Comme cela a été montré plus haut, l'extraction du prénom est également essentielle pour identifier les autres informations du `name` par élimination. La reconstruction des noms a lieu en trois étapes :

— D'abord, un prénom, abrégé ou complet, est extrait.

— Ensuite, le type de prénom est identifié. Pour cet algorithme, trois types de prénoms existent : les prénoms complets (composés ou non), les prénoms abrégés non composés et les prénoms abrégés composés. Le choix de faire une distinction entre les deux catégories s'explique par la différence dans le processus de reconstruction

— Enfin, l'algorithme cherche à reconstruire le prénom à l'aide de tables de conversions.

La première question qui se pose est : qu'est-ce qui constitue un prénom ? Celui-ci correspond à un motif (un ou plusieurs noms propres), mais il est identifié par sa position dans la phase : le prénom est la première information contenue entre parenthèses – cela vaut autant pour les noms de personnes nobles que pour les autres. Pour être certain de cibler uniquement le prénom, cette étape n'a donc lieu que dans les `name` contenant des parenthèses. Bien qu'un motif « prénom » puisse être identifié et que la position du prénom soit fixe, il est en fait assez difficile de modéliser ce motif de façon précise. Il faut définir un ou plusieurs motifs qui correspondent à tous les cas de figures, à toutes les formes que peuvent prendre un prénom. Voici quelques exemples pour mieux cibler cette difficulté (les prénoms sont en gras) :

— « Sobieski (**Thérèse-Cunégonde**) » : un cas classique : prénom composé où les deux noms sont complets et séparés par des tirets.

— « Zimmermann (**P.-J.-G.**) » : une autre forme simple : un nom composé où seules restent les initiales, écrites en majuscules, terminées par des points et séparées par des tirets.

— « DUBOIS-FONTANELLE (**J. Gaspard**) » : une forme un peu plus complexe : le nom est partiellement abrégé, avec une initiale et un nom complet ; cependant, il

14. Par exemple, voici une recherche sur Jean-Jacques Ampère. Si l'on recherche « J J Ampère », les résultats obtenus sont incorrects. En recherchant « Jean Jacques Ampère », le résultat obtenu est le bon.

n'y a pas de tiret qui sépare les deux prénoms, et il faut donc que les deux parties soient détectées toutes les deux dans le même prénom (sans quoi, il existe un risque de se retrouver avec des prénoms partiels).

— « DESCHAMPS (**Jh Fr. L.**) » : pour finir, une forme qui mélange les complexités : trois prénoms abrégés, mais qui ne contiennent pas qu'une initiale ; il ne sont pas tous terminés par des points et ne sont pas séparés par des tirets.

Cette variété tient autant du bruit dans les données que de différentes formes de notation dans le corpus. Elle complexifie en tout cas la définition, d'un point de vue formel, de ce qui constitue un prénom ; il est notamment difficile d'extraire l'intégralité d'un prénom composé, puisqu'il faut faire en sorte que les différents noms ou initiales soient reconnus comme faisant partie du même nom, qu'ils soient séparés par des tirets ou non. Dans l'abstrait.

— Un nom composé abrégé est une série de majuscules suivies d'autres lettres et/ou de points ; il doit contenir au moins un point et un tiret (un motif plus précis risque de ne pas identifier tous les cas de figure ; un motif plus généraliste risque d'inclure trop de bruit).

— Un nom simple abrégé correspond à une majuscule, suivie ou non d'autres lettres et terminée par un point.

— Un nom complet est une série de majuscules suivies de minuscules, séparées ou non par des tirets.

Identifier ces motifs « dans l'abstrait » est une étape importante, qui permet de mettre en place une implémentation technique. Cependant, il est impossible de construire un motif unique correspondant à tous les cas de figures. La décision a donc été d'accepter cette variété et ce bruit et de créer une série de motifs sous la forme d'expressions régulières, qui permettent de cibler différents cas de figure. [15]. À partir d'une connaissance du corpus et de sa structure, il est donc possible d'identifier tous les prénoms dans le texte, et d'automatiser cette identification à l'aide de langages de programmation détectant des motifs.

Une fois les prénoms identifiés, il s'agit de reconstruire ceux qui sont abrégés. Comme pour le reste du processus d'extraction d'informations, cette étape s'appuie fortement sur l'usage de tables de conversion. Le principe général est simple : des tables de conversion sous la forme de dictionnaires contiennent en clés des formes abrégées de noms, et en valeurs les formes complètes. Si un nom extrait se retrouve en clé de ces tables, alors il est remplacé par sa version complète. Malheureusement, l'implémentation technique n'est pas aussi facile. En regardant les tables de conversions utilisées (D.8, D.9), il apparaît

15. Les trois fonctions permettant l'extraction de prénoms sont visibles ici ; D.10 permet d'identifier les noms composés abrégés, D.11 identifie des noms abrégés non composés et D.12 se consacre aux noms complets

que celles-ci sont assez courtes, et ne contiennent pas une grande variété de noms. Le problème avec l'usage de tables de conversion, c'est d'abord qu'elles doivent être faites à la main, et qu'il n'est pas possible de prendre toutes les abréviations possibles en compte ; mais le véritable problème est qu'une seule abréviation peut faire référence à plusieurs noms. C'est malheureusement le cas des abréviations les plus banales : « J. » peut à la fois signifier « Jean », « Jeanne », « Joséphique »... Le choix a donc été fait de ne retenir que les abréviations « évidentes » pour éviter d'introduire du bruit supplémentaire. De plus, quand un nom est modifié par l'algorithme de reconstruction, cela est indiqué à l'aide de la clé `rebuilt` du dictionnaire utilisé pour lancer des requêtes (4.11). Du fait de cette relative pauvreté des données, l'approche est ici totalement prédictive : il est impossible d'être certain d'obtenir le bon nom complet à partir de son abréviation ; il est seulement possible de prédire que le prénom reconstruit sera conforme au vrai prénom (tel qu'il est écrit sur *Wikidata*). Par son fonctionnement modulaire, qui s'adapte aux différents cas de figure, l'algorithme de reconstitution des prénoms cherche à maximiser cette certitude.

L'algorithme de reconstruction des prénoms a été conçu pour chercher à palier à cette relative pauvreté des données de conversion, surtout dans le cas des noms composés abrégés. Ceux-ci posent en effet un problème au delà de leur simple extraction : presque toutes les combinaisons de prénoms sont possibles. L'algorithme cherche donc à reconstruire un prénom composé en plusieurs étapes, d'abord en utilisant des tables de conversion dédiées aux noms composés, puis en utilisant des tables ne comportant que des noms composés. Pour plus de clarté, ce processus sera décrit à partir de l'exemple de « J.-P.-Ch. », à partir duquel il faut arriver à « Jean-Pierre-Charles ». La reconstruction d'un prénom composé commence par une tentative d'alignement complet un prénom composé abrégé présent dans les tables de conversion : un remplacement ne sera fait que si prénom correspond parfaitement aux données présentes dans la table visible en (D.8). Si il y a alignement complet, alors l'extraction du prénom s'arrête ici. Dans notre exemple, ce n'est pas le cas. Il faut tout de même tenter de reconstruire un prénom, ne serait-ce que partiel. Pour minimiser le risque d'erreur, cette reconstruction partielle se fait en deux temps. Dans un premier temps, l'algorithme cherche à associer une partie des initiales extraites avec des clés de la table consacrée aux noms composés. Dans l'exemple, « J.-P. » serait transformé en « Jean-Pierre », puisque « j p » se trouve dans la table de conversion. Si l'intégralité du nom abrégé est reconstruite à l'issue de cette étape, l'algorithme prend fin. Dans notre exemple, il reste à transformer le « Ch. » en « Charles ». L'algorithme essaye donc ensuite d'aligner les initiales restantes avec des clés de la table consacrée aux noms non composés (D.9). « ch » étant présent dans cette table, le dernier nom abrégé est remplacé par un nom complet : un nom a donc été entièrement reconstruit à partir de son abréviation. Tout au long de ce processus, un système permet de suivre quelles initiales ont été remplacées et lesquelles restent à être traitées, pour éviter de traduire deux fois une seule initiale. À la fin de ce processus, toutes les initiales qui n'ont pas été

remplacées sont supprimées : si les abréviations étaient conservées, cela compromettrait l'obtention de résultats sur l'API. À l'inverse, si seulement une partie des prénoms d'une personne sont extraits, mais que les abréviations sont supprimées, la possibilité d'obtenir des résultats sur l'API augmente.

L'alignement d'un prénom abrégé non-composé avec son équivalent complet est bien moins complexe : l'abréviation d'un prénom non-composé étant plus simple, elle ne permet pas de concevoir d'algorithmes réactifs qui puissent adopter différents comportements en fonction des résultats obtenus. Une fois un prénom abrégé extrait, l'algorithme cherche à l'aligner avec son équivalent complet présent dans la table D.9. Si un prénom complet est reconstruit, celui-ci est utilisé pour lancer des requêtes sur l'API de *Wikidata*. Sinon, l'abréviation est supprimée pour ne pas mettre en danger l'obtention de résultats.

Parmi les trois types de prénoms détectés – prénom composé abrégé, prénon non-composé abrégé et prénom complet –, il ne reste donc que les prénoms complets. En théorie, ceux-ci ne demandent pas de retraitement. Cependant, des erreurs peuvent subsister dans les données utilisées. Il est possible que les « marqueurs » d'initiales et d'abréviations (le point à la fin d'un mot commenant par une majuscule) soient absents des documents sources ; le point étant un caractère peu visible, il est également possible que celui-ci soit présent dans les documents d'origine mais qu'il ne soit pas reconnu lors de l'extraction du texte par océrisation. Quelle que soit l'origine du problème, cela mène à des situations comme cela : `<name type="author">MALET (Cl-François)</name>`. Ici, « Cl » est une abréviation ; la structure du nom est cependant celle d'un nom propre composé. La détection de motifs ne pouvant s'appuyer que sur la forme du texte, et non sur sa signification, ce nom est reconnu comme un nom complet. Lorsqu'un prénom est identifié comme étant non abrégé, un une étape supplémentaire de vérification a donc lieu : l'algorithme cherche à associer le ou les prénoms avec des prénoms abrégés présents dans les tables de conversions. Si ils se trouvent effectivement dans des tables, alors ils sont remplacés par leur équivalent complet. Sinon, par élimination, les prénoms sont considérés comme n'étant pas abrégés et sont utilisés pour lancer des requêtes sur l'API.

Ce processus de reconstitution des prénoms permet d'augmenter la qualité des données en cherchant à pallier à leur plus grand écueil. Bien que ce processus soit totalement prédictif et qu'il vienne avec un certain risque d'erreur, son rôle est essentiel dans l'obtention de résultats valides sur le moteur de recherche de l'API *Wikidata*, surtout en l'absence d'informations biographiques. Cela apparaît dans les tests menés, et dont le résultat est visible en annexes (B.1). Lorsqu'une requête est lancée seulement sur le prénom et le nom de famille usuel, l'identifiant *Wikidata* récupéré est valide dans 48% des cas en reconstituant les prénoms. Si des requêtes sont lancées sur les noms de famille et les prénoms non reconstitués uniquement, ce taux de réussite tombe à 42%. Cette différence est non négligeable lorsque l'on travaille sur 82000 entrées différentes à requêter. Ainsi, en utilisant une approche modulable et adaptée au corpus traité, il est possible de pallier à

certaines absences dans les données disponibles. S'il n'y a pas toujours assez d'informations pour s'assurer que l'identifiant récupéré sur *Wikidata* sera le bon, reconstituer les prénoms permet d'augmenter leur précision des données, et donc d'obtenir de meilleurs résultats. Comme cela a été dit plus tôt, cette approche est totalement prédictive, et il est difficile de garantir qu'un prénom reconstruit sera correct. Cette incertitude ne doit pas être oubliée dans la constitution du programme chargé de lancer les requêtes en tant que tel : toute opération d'extraction et de reconstitution des données risque d'introduire du bruit. Il est donc important de lancer les recherches sur l'`API` de manière, là encore, modulaire, en multipliant le nombre de recherches pour chercher à pallier l'imperfection des données.

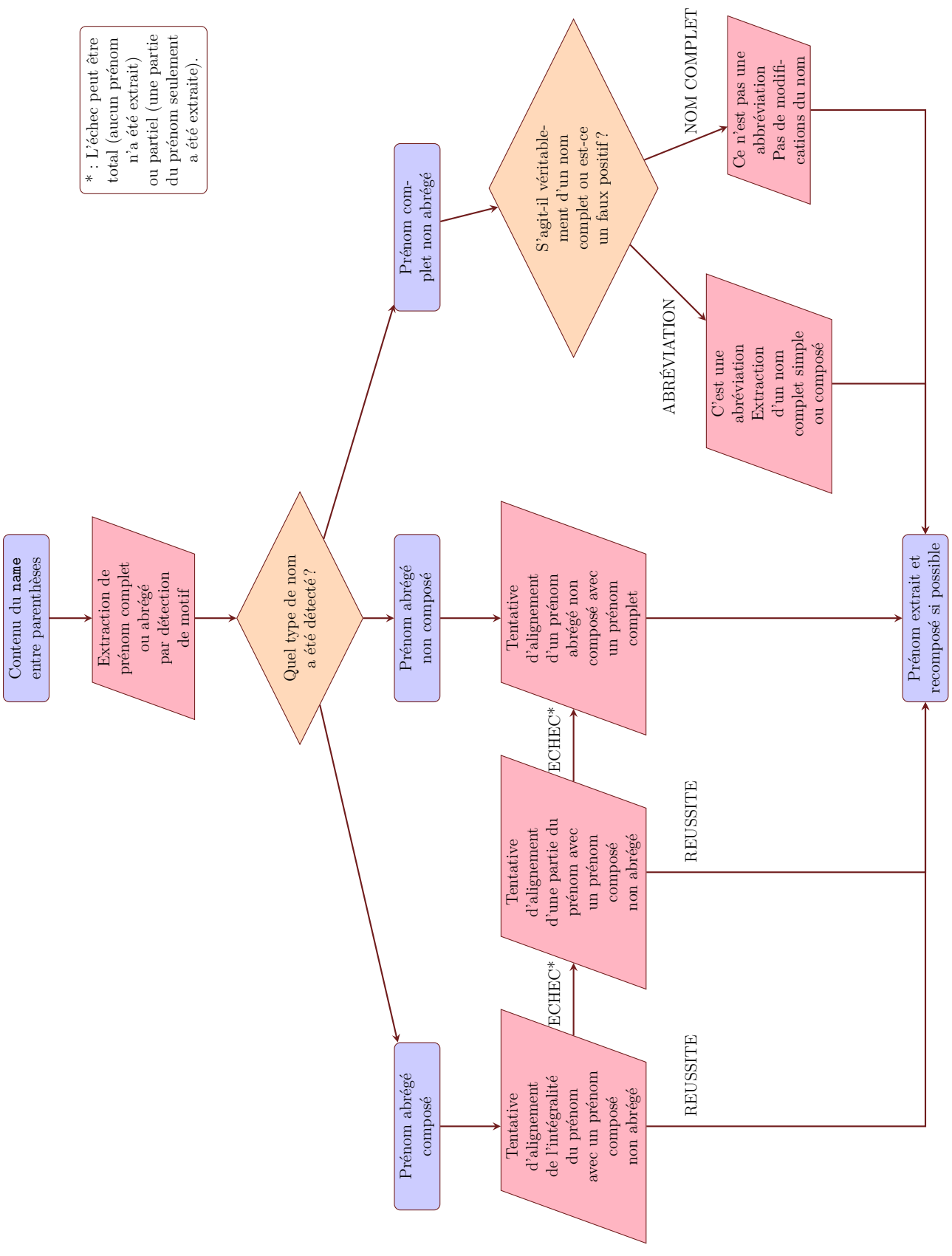


FIGURE 4.7 – Représentation graphique du processus d'extraction et de reconstitution d'un nom à partir de son abréviation

4.2.4 Extraire des informations biographiques du `trait`

Si les informations nominatives sont déterminantes pour récupérer un identifiant valide, elles sont souvent incomplètes ; la manière dont les informations y figurent ne correspond pas non plus toujours à ce qui est disponible sur *Wikidata*. En plus du processus d'extraction d'informations nominatives issu du `name` présenté ci-dessus, des informations biographiques sont extraites dans le `trait`. Pour cette étape encore, l'extraction s'appuie sur une détection de motifs augmentée par l'usage de tables de conversion. Deux informations sont récupérées : les dates de vie et de mort – information cruciale car présentant un risque d'erreur très faible – et la fonction occupée par une personne – ce qui permet d'augmenter la certitude que l'identifiant *Wikidata* récupéré pour une entrée de catalogue est valide.

Identifier les dates de vie et de mort d'une personne

Comparativement à l'extraction d'informations nominatives, l'extraction du nom d'une personne par détection de motifs semble assez simple. Là où un nom peut prendre de nombreuses formes et structures différentes, l'expression régulière correspondant à une date correspond à une suite de quatre chiffres, soit :

```
\d{4}
```

S'il est facile de détecter une date, il est moins évident d'être certain que l'année extraite correspond à la naissance ou à la mort d'une personne. En effet, d'autres dates sont souvent présentes dans le `trait`, comme cela apparaît dans l'exemple ci-desous (4.16). Se contenter d'une extraction de toutes les dates du `trait` risquerait de faire exploser la quantité de bruit dans les données utilisées pour lancer des recherches sur l'`API` *Wikidata*. Comme avec l'extraction d'informations du `name`, détecter des motifs dans le texte ne permet pas d'attribuer une signification à ces motifs – il ne suffit pas de repérer une date pour que cette date correspond à la naissance ou à la mort d'une personne.

Pour comprendre le sens des dates repérées dans le texte, il est encore une fois nécessaire de replacer cette date dans son contexte en s'appuyant sur la nature semi-structurée des documents. En effet, les `trait` ayant tous une structure équivalente et utilisant un vocabulaire limité, il est possible de distinguer les dates pertinentes des autres. Les dates de naissance et de mort sont toujours des compléments dans les phrases, comme cela apparaît dans les différents exemples présentés ci-dessus (4.16). Dans l'exemple 5, par exemple, la date de naissance est un complément de « né » et la date de décès de « fusillé ». C'est donc en identifiant ce à quoi une date se rapporte qu'il est possible de distinguer les dates pertinentes des autres. En général, une date naissance est précédée par « né à, née à ». Le vocabulaire utilisé avec la date de décès d'une personne, quoique plus coloré, est la encore assez limité : « décapité.e, assassiné.e, fusillé.e, guillotiné.e, tué.e ». Ce vocabulaire

```
<!-- exemple 1 -->
<trait>
    <p>maire (en 1848) du XIIe arrondissement, médecin, sous-directeur
    ↪ de l'hôpital de la Salpétrière.</p>
</trait>

<!-- exemple 2 -->
<trait>
    <p>célèbre maréchal de France, vainqueur d'Alger en 1830, né en
    ↪ 1773, mort en 1846</p>
</trait>

<!-- exemple 3 -->
<trait>
    <p>brave général de l'armée de l'Est pendant la guerre de 1870, né à
    ↪ Sarreguemines en 1840, mort en 1876</p>
</trait>

<!-- exemple 4 -->
<trait>
    <p>célèbre révolutionnaire, membre de la Commune de Paris, né en
    ↪ 1838, tué à Rueil le 3 avril 1871.</p>
</trait>

<!-- exemple 5 -->
<trait>
    <p>célèbre jurisconsulte, ministre et sénateur, un des ôtages de la
    ↪ Commune, né à Valence (Drôme) en 1804, fusillé avec l'archevêque
    ↪ de Paris le 27 mai 1871</p>
</trait>
```

Code source 4.16 – Exemples de `trait` contenant des dates autres que les dates de naissance et de mort

étant relativement limité, il n'est pas nécessaire d'utiliser de tables de conversions. Dans le motif permettant d'extraire une date, il suffit simplement de prendre en compte le contexte de celle-ci. Le motif à rechercher correspond donc à :

`Mot indiquant une naissance ou un décès -- autres informations (lieu de naissance...) -- date`

L'implémentation de ce motif sous la forme d'expressions régulières correspond donc à :

pour les dates de naissance : `(^|\s|,|)[Nn](.|\s|ée?).+?(?=\d{4})\d{4}`
pour les dates de decès : `(^|\s|,|)((M|m)|[Mm](\s|orte?)).+?(?=\d{4})\d{4}`

et : `(^|\s|,|)([Dd]écap|[Aa]ssa|[Tt]uée?|[Ff]usi|[Gg]uil).+?(?=\d{4})\d{4}`

Si ce motif est détecté, il est certain que la date est une date de naissance ou de décès. Il suffit alors d'extraire la date de son contexte. Ainsi, sur des documents semi-structurés, il est possible, à l'aide de détection de motifs, de resituer des éléments dans leur contexte afin de produire des données propres et réutilisables.

4.2.5 Identifier l'occupation d'une personne

Identifier et extraire l'occupation d'une personne s'appuie également sur de la détection de motifs, cette fois-ci augmentée de tables de conversion : il existe une grande variété dans les différents types de métiers occupés par une personne ; de plus, un métier tel qu'il est décrit dans un catalogue n'a pas nécessairement d'équivalent sur *Wikidata*. Il est donc nécessaire de normaliser les données extraites.

Le processus d'extraction d'informations à l'aide de tables est assez similaire à l'extraction de titres de noblesse, par exemple : pour chaque `trait` présent dans les catalogues, l'algorithme vérifie si un terme correspond à une occupation à l'aide de la table D.1. Si c'est le cas, l'occupation telle qu'elle est mentionnée dans les catalogues est remplacée par son équivalent normalisé. Celui-ci n'est pas seulement une traduction du terme tel qu'il est présent dans le `trait` : les occupations ou fonctions sont remplacées par une version normalisée, qui permette d'obtenir des résultats sur le moteur de recherche de *Wikidata*. Par exemple, le terme « maréchal » sera remplacé par « marshall » (sa traduction anglaise), tandis que des grades militaires moins élevés (« capitaine », « commandant ») seront remplacés par le terme générique « militaire ». L'extraction d'informations n'étant pas viser à obtenir des données définitives, il n'est pas nécessaire de préserver les termes et les catégories utilisées dans les catalogues.

Cette extraction d'informations « à l'aveugle » – qui se soucie uniquement des mots utilisés et non du contexte dans lequel il le sont – présente deux problèmes, visibles dans les exemples ci-dessous (4.17).

— Dans le premier exemple, une personne est à la fois désignée comme étant « sculpteur » et « auteur ». À l'aide de la table de conversion, les deux termes sont extraits et normalisés, ce qui donne « sculptor writer »[16]. Non seulement la transformation est fausse (il ne s'agit pas d'un écrivain), mais elle rend inutile l'extraction d'informations : les deux termes se contredisent et ne permettront pas d'obtenir de résultats sur *Wikidata*. Ce problème est récurrent, puisque des personnes ont souvent écrit ou réalisé une œuvre sans qu'elles soient connues comme écrivaines – comme cela apparaît dans le deuxième exemple, où le personnage est général, mais a écrit des mémoires. Il faut donc déterminer laquelle de ces deux fonctions est à retenir pour faire des recherches sur l'`API` *Wikidata*.

16. « sculpeur écrivain »

— Au premier problème de l'extraction de plusieurs fonctions s'ajoute une deuxième difficulté : les occupations extraites ne se rattachent pas nécessairement à la personne nommée dans le `name`. C'est le cas dans le troisième exemple : « empereur » est présent dans la table de conversion – contrairement à son équivalent féminin, qui ne permet pas souvent d'obtenir de résultats. La fonction extraite sera donc « empereur / emperor », ce qui fausse les données extraites : c'est la femme de l'empereur qui est décrite dans cette entrée. Il est donc nécessaire de reconnaître lorsque, dans le `trait`, une fonction ne se rattache pas à l'auteur.ice du manuscrit.

```
<!-- exemple 1 -->
<trait>
    <p>célèbre sculpteur, auteur de la statue de la Résistance, érigée à
    ↪ Dijon en 1875 et enlevée par ordre du gouvernement, né à Nuits
    ↪ (Côte-d'Or) en 1815, mort en 1876</p>
</trait>

<!-- exemple 2 -->
<trait>
    <p>célèbre général auteur de Mémoires, qui ont eu un grand succès,
    ↪ n. 1782, m. 1854.</p>
</trait>

<!-- exemple 3 -->
<trait>
    <p>impératrice des Romains, reine de Hongrie, fille de
    ↪ Charles-Quint, femme de l'empereur Maximilien II, morte en
    ↪ 1603</p>
</trait>
```

Code source 4.17 – Difficultés liées à l'extraction de fonctions du `trait`

Les deux problèmes trouvent des solutions différentes. Le premier problème naît de la différence entre les informations présentes dans les catalogues et celles qui sont nécessaires pour l'extraction d'informations de *Wikidata*. Dans un texte fait pour être lu par des humains, il n'y a aucun problème à mettre bout à bout plusieurs occupations, fonctions ou faits notables à propos d'une personne. Un.e lecteur.ice est capable de comprendre la polysémie des mots (« auteur », par exemple) et la relation des différentes fonctions entre elles. Dans le cas d'un catalogue, qui est fait pour vendre, il est normal que les descriptions de personnes soient mélioratives : la multiplication des hauts faits à propos d'une personne met en avant son statut, et justifie donc la valeur du manucrit vendu. Les techniques de traitement automatique utilisée ne comprennent pas la polysémie. Comme cela a été dit plus haut, le moteur de recherche de *Wikidata* fonctionne mieux avec des termes spécifiques ; il est moins efficace lorsque le nom d'une personne est recherché avec

plusieurs fonctions en même temps. Il est donc nécessaire de déterminer l'occupation principale d'une personne. Après l'extraction des fonctions, les différents termes extraits sont comparés entre eux afin de déterminer le terme à retenir. La comparaison se fait entre les mots qui sont cooccurents, c'est-à-dire qui se retrouvent régulièrement ensemble, comme par exemple « marshall » et « military ». Dans ce cas, c'est la fonction la plus importante et la plus spécifique (« marshall ») qui est conservée, et le terme générique est abandonné. Les termes très génériques – comme le « writer » qui est traduit d'« auteur » – sont presque systématiquement abandonnés si d'autres termes ont été extraits : un.e auteur.ice n'est un.e écrivain.e que si ce terme est le seul utilisé pour le ou la décrire. Sinon, la personne est autrice de quelque chose sans que cela soit sa fonction principale.

Le second problème naît de l'algorithme d'extraction d'informations lui-même : la récupération de fonctions ou de métiers est non-contextuelle : si un terme est présent, il est récupéré. Un post-traitement est donc nécessaire afin de déterminer si le terme se rapporte à l'auteur.ice du manuscrit, ou à une autre personne. L'algorithme cherche à voir si le terme est précédé des prépositions « du, de la, des », comme c'est le cas dans le troisième exemple ci-dessus (4.17). Dans ce cas, il est considéré que l'information récupérée n'est pas relative à l'auteur.ice, mais à une tierce personne et elle est donc abandonnée.

La récupération de données biographiques du `trait` marque la fin du processus d'extraction d'informations ; à l'issue de celui-ci, l'ensemble des éléments récupérés sont stockés dans un dictionnaire structuré (4.11), ce qui permet d'attribuer à chaque terme une fonction, une signification spécifique. À partir d'un texte en langage naturel, il est donc possible d'extraire un certain nombre de paramètres à partir desquels lancer des recherches en plein texte. C'est grâce à la nature semi-structurée des entrées – qui utilisent un vocabulaire peu varié dans des phrases structurellement proches les unes des autres – qu'il est possible de faire une extraction d'informations avec un grand degré de précision en s'appuyant uniquement sur de la détection de motifs. Grâce à une bonne connaissance du corpus et à l'aide d'une approche modulaire, qui s'adapte à différents cas de figure, il est possible d'approximer une compréhension « humaine » du texte. L'extraction d'informations n'étant qu'un étape préalable, cette compréhension du texte est uniquement centrée sur les éléments qui sont pertinents pour le moteur de recherche de *Wikidata*. Cette étape d'extraction est cependant prédictive : elle peut inclure du bruit dans des données, les informations extraites peuvent être mal structurées, puisque leur fonction est déterminée principalement en fonction de leur place dans le texte. De ce fait, l'étape suivante se doit de répondre à cette part de bruit possible, à l'aide d'un algorithme qui soit ici encore modulable. L'attribution à chaque terme d'une valeur ou d'une fonction spécifique est mise à profit pour pouvoir construire de façon automatique des recherches intelligentes, qui s'adaptent autant aux données disponibles que à l'outil auquel elles sont destinées : un moteur de recherche.

4.3 Extraire des identifiants *Wikidata*

En utilisant les données extraites à l'étape précédente, des recherches en plein texte sont faites sur *Wikidata* de façon automatique, grâce à l'`API` développée par cette institution. *Wikidata* retourne plusieurs résultats ; il est impossible de vérifier quel résultat est pertinent parmi tous ceux qui sont obtenus, et seul le premier résultat est donc retenu. Ce résultat contient un identifiant *Wikidata*, à partir duquel des requêtes pourront être menées avec `SPARQL` afin de constituer un jeu de données sur les auteur.ice.s figurant dans les catalogues de vente de manuscrits. L'objectif de l'algorithme est alors de chercher à maximiser la probabilité que, avec les données produites à l'étape précédente, le premier résultat retourné par l'`API` soit correct. Pour savoir comment maximiser cette probabilité, il est nécessaire de mieux comprendre la fiabilité des différents paramètres extraits à l'étape précédente. Après avoir étudié l'impact des différents paramètres dans l'obtention d'un résultat valide et présenté le fonctionnement d'une `API`, la manière dont l'algorithme a été constitué sera présentée.

4.3.1 Quantifier l'incertitude : quels sont les paramètres qui permettent d'obtenir le bon résultat ?

Tout au long de la constitution de cet algorithme, des tests ont été menés pour maximiser sa qualité et la probabilité d'obtenir des résultats valides. Le premier test est essentiel à la constitution de l'algorithme : il mesure l'impact de chaque paramètre de recherche (dates de naissance et de décès, titres...) dans l'obtention d'un identifiant valide (B.1).

Ce test a été mené sur un jeu de 200 entrées sélectionnées dans différents catalogues, avec une répartition des les différents types de `name` similaire à celle du jeu de données complet ; de plus, la proportion d'entrées avec des `trait` est la même dans le jeu de test et dans l'ensemble du catalogue. Pour chacune de ces 200 entrées, des données sont extraites du `name` et du `trait` ; ensuite, cinq requêtes sont lancées pour chaque entrée : d'abord une requête avec le nom de famille usuel et le prénom seulement ; ensuite, une requête avec ces deux informations et un paramètre en plus – nom de famille noble, titre de noblesse, dates, occupation. Les cinq requêtes sont lancées, peu importe si les différentes données sont disponibles pour une entrée ou non : l'intérêt est de pouvoir quantifier l'impact de chaque paramètre pour l'ensemble des entrées, et pas seulement pour celles où le paramètre est disponible. Par exemple, si les dates de naissance et de décès permettent une obtention du bon résultat dans 90% des cas mais qu'elles sont disponibles pour 5% entrées, leur impact global sur l'obtention d'un résultat valide est en fait très limité. Comme cela a été dit plus haut, ces cinq requêtes sont lancées deux fois : d'abord avec le nom et le prénom complet reconstitué si besoin, ensuite avec seulement les prénoms non-reconstitués, afin de vérifier

l'utilité de la reconstitution (les résultats sont très positifs, avec une augmentation de performance allant jusqu'à +6%).

Ces tests permettent donc de connaître, au niveau de l'ensemble du jeu de données, les paramètres les plus fiables pour obtenir des résultats valides. Il en ressort que le paramètre le plus efficace est l'occupation, ou le métier d'une personne : celui-ci permet d'obtenir des résultats valides pour 53,1% des entrées (contre 48% en recherchant les noms et prénoms, et 42% en cherchant les noms et uniquement les prénoms non-reconstruits). Ce résultat peut sembler inattendu puisque les fonctions ne sont pas toutes relevées et qu'elles subissent un retraitement, ce qui pourrait créer du bruit additionnel dans les recherches. Cependant, il montre l'importance de la connaissance à la fois de son corpus et de *Wikidata* : c'est en s'adaptant à son moteur de recherche qu'une extraction efficace des fonctions a été rendue possible. Le deuxième paramètre le plus efficace est les dates de naissance et de mort (52,6% de résultats valides) : c'est une information qui n'est pas retraduite, et il n'y a pas de risque de création de bruit additionnel. Suivent les noms de famille noble (45,9% de 40,5%, respectivement avec et sans reconstitution des prénoms). Il est intéressant de remarquer que ce paramètre fait diminuer les performances des recherches (entre 1,5 et 2,1 points de moins qu'en utilisant uniquement prénom et nom de famille usuel). Cette chute s'explique probablement par le fait que, sur *Wikidata*, une personne est rarement nommée à la fois par son nom usuel et par son nom de famille noble. Dans certains cas cependant, rechercher le nom de famille noble à la place du nom usuel peut permettre d'obtenir des résultats qui n'auraient pas été trouvés autrement.

En quantifiant l'impact de chaque paramètre de recherche dans l'obtention d'un résultat, ce test permet de savoir à quelles données se fier et quelles données rechercher en premier sur l'`API` de *Wikidata*. Il permet donc de quantifier la certitude, ou l'incertitude, liée à chaque paramètre, et donc de développer un algorithme de recherche de façon plus intelligente. Mais avant de présenter cet algorithme, il peut être intéressant de revenir sur ce qu'est une `API`, et sur la manière de les utiliser dans un contexte de recherche.

4.3.2 Automatiser l'enrichissement de données via des sources externes : les `API`

Qu'est-ce qu'une `API` ?

Une `API` Web[17] est un protocole qui permet à un programme d'interagir avec un autre dans le cadre d'une architecture client-serveur. Cette architecture est à la base du Web, qui est formé d'un ensemble d'ordinateurs reliés en réseau. Certains ordinateurs sont des hôtes et servent à diffuser du contenu ou des données : ce sont les serveurs. D'autres ordinateurs sont des clients, c'est-à-dire qu'ils consomment ce qui est fourni

17. Il existe également des `API` qui ne sont pas faites pour le Web, et qui permettent par exemple la communication entre deux programmes présents sur un même ordinateur.

par les serveurs [18]. Le client lance des requêtes qui sont traitées par le serveur, qui en retour renvoie une réponse. Ce réseau d'ordinateurs communique grâce à des protocoles, au cœur desquels se trouve le *HyperText Transfer Protocol* (HTTP) [19]. Le rôle de celui-ci est de définir comment un client et un serveur communiquent : il établit la sémantique des requêtes envoyées d'un client à un serveur, et des réponses que ce dernier renvoie en retour. Le protocole HTTP définit donc de façon générale comment deux machines communiquent : lorsqu'un client pose une question, le serveur saura toujours ce que la question signifie ; de la même manière, le client pourra toujours interpréter la réponse du serveur. Cependant, ce protocole ne définit pas ce qui peut être demandé par un client à un serveur ; il ne définit pas non plus précisément le modèle de données utilisé par la réponse du serveur. Des technologies et des protocoles supplémentaires sont donc nécessaires, pour définir le contenu des requêtes et des réponses, en plus de leur sémantique. C'est ici que les `API` prennent leur importance : elles définissent ce qui peut être requêté, le vocabulaire spécifique à utiliser par les clients et les formats de réponse qui seront fournis en retour. En plus de définir un protocole, elles sont des programmes complets côté serveur : lorsqu'elle reçoit une requête d'un client, l'`API` traite celle-ci, construit une réponse en interagissant avec serveur et la renvoie à un client. L'`API` est donc un programme, avec des fonctions qui traitent des données, et un protocole, avec un vocabulaire permettre l'interaction client-serveur. Contrairement au HTTP qui est générique et défini à l'échelle du Web, les `API` sont spécifiques à un projet, à une application (chaque application ayant des besoins différents) ou à un groupe d'applications [20]. Une `API` permet donc l'interaction entre deux types de programmes : le programme dit fournisseur (qui est soit l'`API` en tant que telle, soit un programme côté serveur) et le consommateur (c'est-à-dire, un programme côté client) [21]. D'un point de vue de la conception d'applications, les `API` ont de nombreux avantages. Elles garantissent la sécurité des données : selon les principes REST [22], le client n'accède jamais directement à une base de données dans le serveur, mais seulement à une représentation des données produite à la demande ; puisque les données enregistrées sont distinctes des données communiquées, il est également possible de modifier la base de données derrière l'application sans pour autant devoir modifier l'intégralité de l'`API` ; de la même manière, le programme client n'a pas besoin d'être mis à jour à chaque mise à jour côté serveur : il doit seulement être modifié si l'`API` est modifiée.

DEPLACER EN INTRO DE PARTIE ?????????????????????? Si l'élaboration d'une `API` est intéressant d'un point de vue de développement, d'architecture Web et d'interaction entre machines, l'utilisation d'`API` dans un contexte de recherche en humanités peut

18. Dans les faits, une seule machine peut à la fois être client et serveur.

19. **fielding_architectural_2000**.

20. Il existe cependant des standards dans l'architecture des `API`, comme le REST, défini par R. Fielding (**fielding_architectural_2000**)

21. Pour une représentation graphique de la communication avec une `API` dans une architecture client-serveur, voir la figure 4.8

22. **fielding_architectural_2000**.

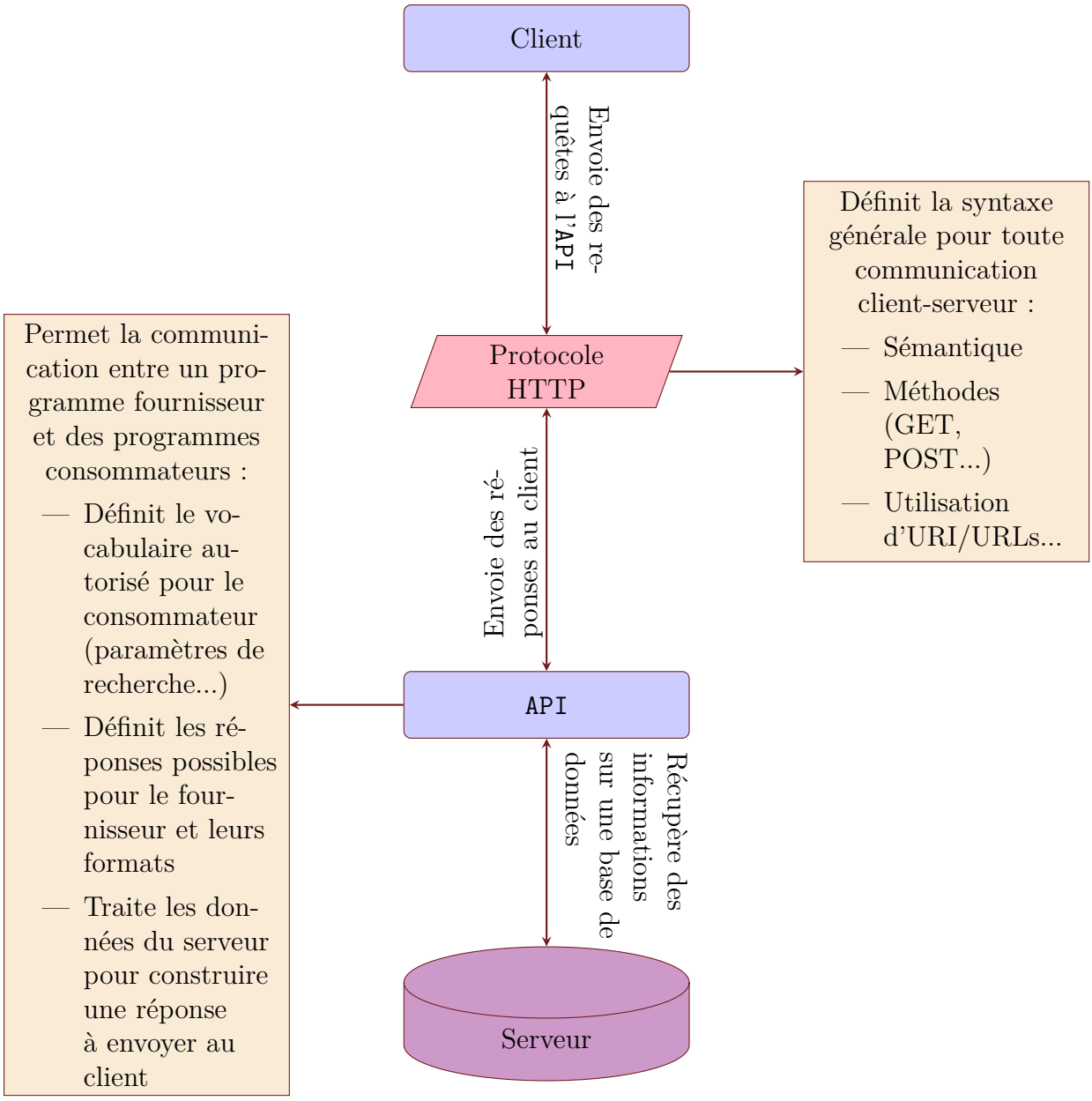


FIGURE 4.8 – L'interaction client-serveur au travers d'une API

être très fructueuse. Une API permet la communication de données brutes (c'est-à-dire, d'informations « à l'état pur », sans mise en page ou interface graphique) dans des formats documentés. Consommer une API [23] permet donc d'obtenir de façon automatisée des données déjà structurées, dans un format qui soit compréhensible par une personne humaine autant que par une machine. L'utilisation d'une telle technologie est donc totalement adaptée à l'alignement avec *Wikidata*. Concrètement, en utilisant l'API de *Wikidata*, il est possible de lancer des recherches en plein texte sur le moteur de recherche de cette base de données de façon automatique. Même si les données obtenues doivent être ensuite corrigées, en écrivant un programme pour consommer une API, on s'épargne le travail de

23. C'est-à-dire récupérer des données

faire manuellement plus de 80000 recherches.

Consommer une `API` : le cas de l'application *MediaWiki*

Wikidata et les projets qui y sont affiliés (*Wikipedia*, *Wikimedia* et autres projets de la *Wikimedia Foundation*) peuvent être consultés en ligne, par des utilisateur.ice.s humain.e.s, au travers d'un site web traditionnel. Il faut cependant distinguer le site web des données : le site n'est qu'une interface de présentation. Les projets de la *Wikimedia Foundation* existent avant tout sous la forme d'un ensemble de bases de données (E.1), qui sont accessibles au public via le logiciel *open source* MediaWiki [24]. Ce logiciel permet la diffusion des données aux utilisateur.ice.s via une interface graphique traditionnelle ; mais il est également possible de récupérer et d'envoyer des données de la *Wikimedia Foundation* via l'`API` de MediaWiki.

Une `API` web étant un équivalent d'une application Web conçue pour une machine, la manière d'accéder aux données est assez similaire à celle utilisée pour accéder à un site Web traditionnel : la requête passée par le client prend la forme d'un *Uniform Resource Locator* (URL) [25] analogue à celui d'une page web traditionnelle. L'URL contient la plupart des informations nécessaires pour que l'`API` comprenne la requête (interprète ce que le client demande) et construise une réponse (dans notre cas, renvoie des données). Ci-dessous, deux URL correspondant à une recherche en plein texte pour « Virginia Woolf ».

—

Recherche sur le site *Wikidata* :
`https://www.wikidata.org/w/index.php?search=virginia+woolf`

Recherche sur l'`API` *Wikidata* :
`https://www.wikidata.org/w/api.php?action=query&list=search&srsearch=virginia+woolf`

Les deux URL posent la même question ; cependant, le site « pour humain.e.s » est indiqué par `index.php?` ; `api.php?` indique que le deuxième URL renvoie à l'`API` et contient une suite de paramètres de recherches séparés par le caractère `&` :

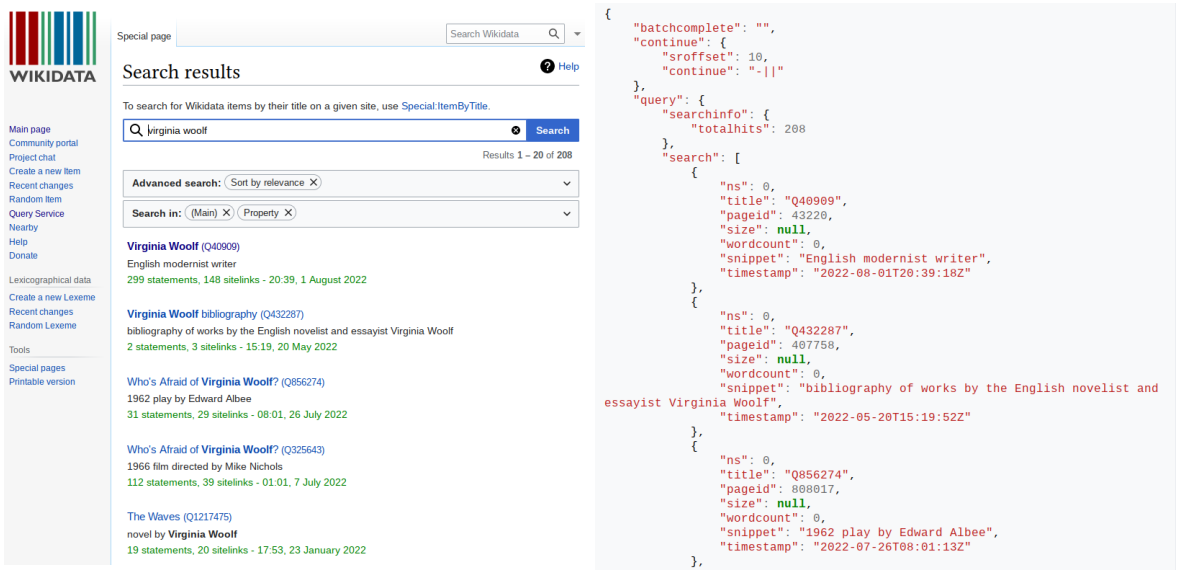
— `action=query` : l'action effectuée par cette URL est une requête demandant des données au serveur.

— `list=search` : il s'agit d'une recherche plein texte.

— `srsearch=virginia+woolf` : rechercher les pages qui contiennent « Virginia Woolf ».

24. **noauthor_wikimedia_2022**.

25. Dans les faits, dans le protocole HTTP les URL sont nécessaires à une requête, mais ne sont qu'une partie de la requête : elle est également composée d'un en-tête et d'une méthode, c'est-à-dire d'une action à faire avec cet URL (demander des données, en envoyer, en supprimer...).



(a) Résultats retournés par le site traditionnel

(b) Résultats retournés par l'`API`

FIGURE 4.9 – Résultats retournés par *Wikidata* pour la recherche « Virginia Woolf »

Enfin, les données récupérées sont les mêmes, mais dans deux formats très différents (4.9). Dans l'image de gauche (4.9a), les résultats sont visibles avec une interface graphique complète, comme sur n'importe quel site web. À droite cependant (4.9b), les résultats sont présentés sous la forme de données structurées au format `JSON` (format analogue à un dictionnaire). Ce résultat contient à la fois les résultats à proprement parler (en dessous de « Search ») et des informations descriptives. Par exemple, `query/searchinfo/totalhits` retourne le nombre de résultats pertinents, soit 208 pages.

Cette brève présentation de la manière de consommer concrètement une `API` indique également le protocole à suivre pour l'alignement des `name` avec des entités *Wikidata* : il s'agit de concevoir un algorithme qui, à partir des données extraites du `name` et du `trait`, construise des URL pour lancer des recherches en plein texte sur *Wikidata*. Cela revient à associer au paramètre de l'URL `srsearch` la chaîne de caractère qui doit être recherchée sur la base de données de *Wikidata*. Ensuite, l'algorithme doit transmettre à l'`API` ces , interpréter le `JSON` renvoyé par *Wikidata* et enregistrer ces réponses dans un fichier. Étant donné que l'alignement avec *Wikidata* prend plusieurs dizaines d'heures et demande des performances élevées, des fichiers de log annexes sont créés en cours d'utilisation pour optimiser l'algorithme.

4.3.3 Présentation générale

L'algorithme d'alignement avec *Wikidata* construit des URL et lance des recherches en plein texte sur l'`API` *Wikidata*. De par son fonctionnement, il multiplie les requêtes. Puisqu'il est impossible de vérifier en cours d'exécution si la réponse retournée par l'`API` contient un résultat valide, il est nécessaire d'organiser l'algorithme afin de maximiser

la probabilité d'obtenir un résultat valide. C'est pourquoi les recherches sont faites en commençant par les celles qui ont la plus grande certitude d'obtenir des résultats valides et en terminant par les plus incertaines. Dès qu'une réponse est obtenue, l'algorithme s'arrête et l'entrée de catalogue suivante est traitée. Comme cela apparaît dans la figure 4.10, il a par conséquent un fonctionnement à plusieurs étapes.

La première étape est la même quelles que soient les informations contenues dans le `name` : l'ensemble des paramètres[26] extraits lors de l'étape précédente sont mis bout à bout afin de lancer une recherche. Cela permet que la recherche soit la plus précise possible, afin de maximiser les possibilités d'obtenir un résultat valide.

Si la réponse de l'`API` ne contient pas de résultat, alors les données disponibles sont étudiées afin d'adopter le bon comportement. Comme cela a été dit, différents processus ont été utilisés pour extraire des données du `name` et du `trait` ; il est donc logique de ne pas traiter tous les dictionnaires de données de la même manière.

Si l'entrée concerne une personne noble (c'est-à-dire, si le dictionnaire de données contient un nom de famille noble), alors il y a un possible conflit dans les données extraites : si le nom de famille usuel est recherché en même temps que le nom de famille noble sur *Wikidata*, il arrive qu'aucun résultat ne soit retourné. Dans ce cas, une première requête est lancée en supprimant le nom de famille noble ; si aucun résultat n'est obtenu, celui-ci est rajouté et c'est le nom de famille usuel qui est enlevé. Pour finir, une autre recherche est lancée sans le prénom. Si à ce stade aucun résultat n'a été obtenu, alors un algorithme de recherches soustractives est lancé ; celui-ci est décrit plus bas.

Le deuxième cas de figure concerne la reconstitution des prénoms. Bien que celle-ci augmente la proportion de résultats valides, un prénom mal reconstitué risque de retourner du silence. C'est pourquoi une clé `rebuilt` a été ajoutée au dictionnaire qui contient les données extraites : cette clé permet d'indiquer qu'un nom a été reconstitué. Dans ce cas, une nouvelle recherche est lancée sans le prénom. Si aucun résultat n'est obtenu, alors c'est l'algorithme de recherches soustractives qui s'exécute.

Le troisième cas de figure concerne la présence de plusieurs paramètres de recherche disponibles – en langage technique, de plusieurs valeurs non-nulles dans le dictionnaire. Dans ce cas, c'est directement l'algorithme soustractif qui s'exécute. Celui-ci a été conçu à partir du constat que, d'un côté, les données contiennent inévitablement de bruit ; de l'autre, l'extraction de données risque elle aussi d'introduire du bruit. Cette logique soustractive cherche donc à supprimer le bruit qui a pu s'introduire. Afin d'éviter d'être trop « brut », l'algorithme procède en séparant les différents types de bruit possibles. D'abord, il arrive souvent qu'un chiffre soit mal reconnu lors de l'océrisation, et il peut donc y avoir des erreurs dans les dates. Deux dates peuvent être extraites (naissance et décès) ; il y a cependant peu de chances d'erreur sur les deux dates. Par conséquent, une

26. Dans ce qui suit, le terme « paramètre » se réfère aux clés du dictionnaire d'informations structurées, et non aux paramètres de l'`API` présentées ci-dessus (`srsearch`...).

des deux dates est supprimée et la recherche est lancée ; si aucun résultat n'est obtenu, c'est l'autre date qui est supprimée. La seconde source de bruit provient des prénoms : comme cela a été dit, au XIX$^{\text{ème}}$ s., les prénoms étrangers sont régulièrement traduits ; il arrive souvent que les prénoms composés ne soient pas dans le même ordre que sur *Wikidata*, ou qu'une personne soit nommée par son deuxième prénom – peut-être parce que les catalogues utilisent des noms d'usage alors que *Wikidata* s'en tient aux noms civils. Pour ces raisons, le prénom est retiré de la chaîne de caractères à rechercher avant de faire une requête sur l'`API`. Bien que les risques liés aux dates et aux prénoms ont été corrigés (autant que possible), il est possible que l'auteur.ice n'ait pas encore été aligné avec une entité *Wikidata*. À ce stade, tous les risques spécifiques ont été traités. L'algorithme relance alors des recherches en retirant à chaque fois un paramètre différent, jusqu'à ce qu'un identifiant ait été extrait de *Wikidata* ou que toutes les permutations possibles aient été faites.

Cette série de construction d'URL cherche à minimiser les redondances ; cependant, il est possible qu'une même chaîne de caractères puisse être extraite à différentes étapes. C'est pourquoi des fichiers de log sont créés en cours d'utilisation : ils servent à sauvegarder les chaînes de caractères qui ont déjà été recherchées et les résultats qui leur sont associées. Avant de lancer une requête sur l'`API`, l'algorithme vérifie donc si la chaîne a déjà été recherchée. Si oui, il récupère le résultat obtenu ; sinon, lance la recherche et sauvegarde le résultat retourné par *Wikidata*.

L'algorithme de recherche sur *Wikidata* est visiblement complexe, puisque son objectif est de minimiser le taux d'erreur possible sur des données variées, ce qui demande d'adopter des comportements différents en fonction des informations disponibles. L'algorithme ayant tendance à retirer de plus en plus de paramètres de recherches en cours d'exécution (les dates, les prénoms...), il est possible que la chaîne recherchée finisse par être très pauvre en informations ; cela permet de s'aligner avec une entité, mais celle-ci risque de ne pas correspondre au `name` ou d'être très générique. Si l'entité recherchée est « Napoléon Bonaparte », mais que suffisamment de paramètres ont été retirés, il est possible que l'algorithme ne recherche que « Bonaparte ». Le résultat obtenu est alors « Charles Lucien Bonaparte ». Il est donc utile d'avoir une mesure de la certitude (ou l'incertitude) de la validité d'un résultat, ne serait-ce que pour accélérer le processus de correction des identifiants (ce qui, sur plus de 82000 entrées, n'est pas une mince affaire). Le parti pris a été, tout simplement, d'établir un score de certitude en fonction du nombre de paramètres utilisés pour lancer une recherche. Ce score a été défini de façon expérimentale, en mesurant la proportion de résultats considérés comme certains et ainsi que le taux d'erreur au sein des résultats certains (c'est-à-dire, le nombre de résultats qui ont dépassé le seuil de certitude mais qui sont en fait erronés). Après plusieurs essais différents, le score de certitude c_r pour un résultat r a été déterminé. Il suit la formule ci dessous :

Étant donnés :

c_r le score de certitude pour un résultat r ;

q_r la chaîne de caractère recherchée pour obtenir le résultat r ;

$\sum param$ la somme des paramètres utilisés dans q_r ;

$d = 1$ si une date est dans q_r ; sinon, $d = 0$;

$p = 1$ si le prénom utilisé dans q_r n'a pas été reconstitué ; sinon, $p = 0$;

$$c_r = \sum param + d + p$$

Un résultat est considéré comme étant « certain » si $c_r >= 4$ ou si des dates sont présentes dans q_r (les chances qu'un alignement soit fait entre un `name` et une entité homonyme ayant les mêmes dates de naissance ou de décès étant très faibles). Sur le jeu de données de test, 32% des entrées atteignent ce seuil de certitude. 23,6% de celles-ci comprennent en fait une erreur, soit 8,5% du jeu de données total (voir le tableau en annexes : B.2). L'objectif de ce score de certitude n'est pas d'être parfaitement exact, mais d'accélérer la relecture des résultats : en acceptant un taux d'erreur total de 8,5%, il est possible ne corriger que les entrées pour lesquelles $c_r < 4$. Seuls les deux tiers du jeu de données restent alors à corriger manuellement.

Cet algorithme cherche à répondre à plusieurs problèmes techniques : il travaille avec des données historiques contenant du bruit et un niveau très inégal d'information. Son fonctionnement vise à maximiser la probabilité d'obtenir un résultat valide, en faisant des requêtes d'abord très spécifiques puis de plus en plus génériques. Cependant, il crée lui-même plusieurs problèmes : son fonctionnement est assez complexe et demande de manipuler plusieurs lourds fichiers en même temps. Un ensemble d'optimisations ont donc été réalisées.

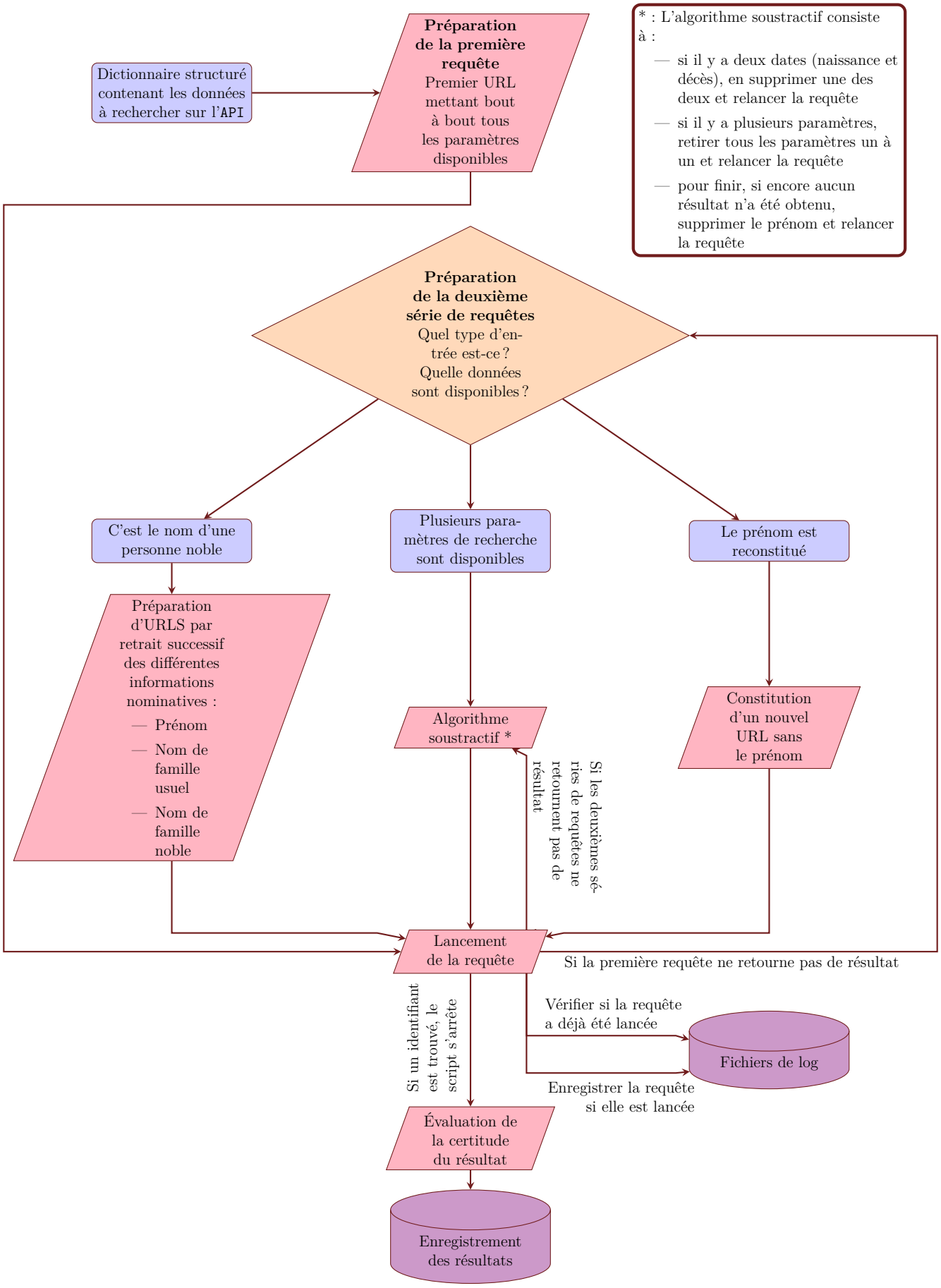


FIGURE 4.10 – Algorithme lançant des recherches en plein texte sur *Wikidata*

4.3.4 Gérer la montée en charge : optimisation et réduction du temps d'exécution

L'intégralité de l'algorithme (contenant extraction d'informations et interaction avec une `API`) a cessé de fonctionner sur mon ordinateur alors qu'à peine 5% des 820000 entrées avaient été traitées. Ce problème technique soulève des difficultés adjacentes à la recherche en humanités numériques : dans ce contexte, des ordintateurs très performants ne sont pas toujours disponibles, et les machines utilisées sont souvent des ordinateurs de bureau traditionnels. De telles situations, la recherche peut parfois faire face à des limitations matérielles. De telles limites sont – bien sûr – un problème ; en obligeant chercheur.euse.s à intégrer des contraintes matérielles dans leurs algorithmes, cette difficulté peut aider à réaliser des programmes plus efficaces, moins gourmands en ressources et en énergie, et donc aider à faire de la recherche plus durable. De telles considérations écologiques sont partagées par toute une communauté de chercheur.euse.s issu.e.s des humanités et défendant des approches « low-tech » [27] ; ces considérations écologiques sortent du champ des humanités et se retrouvent dans des travaux provenant de l'informatique « traditionnelle » et de l'apprentissage machine [28]. L'optimisation de programmes informatique n'est pas seulement une question écologique : c'est également une manière d'écrire de meilleurs programmes, plus performants et efficaces.

Dans sa première version, le script d'alignement avec *Wikidata* avait un fonctionnement apparemment plus « simple », puisqu'il ne prenait pas du tout en compte les questions d'optimisation : pour chaque entrée de catalogue, des informations étaient extraites et les recherches faites sur *Wikidata*. Ce fonctionnement avait deux défauts. D'abord, il demandait que tout le script soit exécuté en une fois – si il y a une erreur imprévue, il est nécessaire de recommencer à nouveau. Ensuite, une interaction avec *Wikidata* était nécessaire pour chaque entrée, peu importe si le nom contenu dans le `name` avait déjà été recherché auparant (alors qu'une même personne est souvent l'auteur.ice de plusieurs manuscrits recensés dans les catalogues) ; l'interaction avec *Wikidata* prenant plusieurs secondes, cela peut grandement ralentir le processus d'alignement avec *Wikidata*. C'est à partir de ces deux problèmes que le processus d'optimisation a commencé.

Le premier problème – l'impossibilité de vérifier quelles entrées de catalogue ont déjà été traitées – a d'abord été corrigé en consultant, à chaque fois qu'un nouvel identifiant était recherché, le fichier de sortie. Celui-ci est un tableur qui contient l'identifiant des entrées, le `name`, le `trait`, l'identifiant récupéré sur *Wikidata* et un extrait de la description

27. Le *Minimal computing* (« Informatique minimale » en français) est un groupe de réflexion créé au sein de *GO ::DH – Global Outlook Digital Humanities*. Il s'intéresse aux manières de rendre l'informatique minimale, dans différents sens du terme : à la fois plus accessible, mais également plus écologique. Plus généralement, le courant du *minimal computing* développe une approche réflexive sur les façon de faire de la recherche durable et libre (au sens de l'*open source*) ; ce groupe s'intéresse également au développement des humanités numériques dans un contexte non-occidental**sayers_minimal_2016**.

28. **strubell_energy_2019**.

associée à cet identifiant. Si cette solution permet de ne pas recommencer à zéro à chaque fois que le script est lancé, elle demande de charger en mémoire un très large fichier `csv` (12,7 mégaoctets). Cette vérification particulièrement gourmande a été remplacée par la création d'un fichier de log annexe en cours d'exécution. Celui-ci contient uniquement les identifiants des différentes entrées de catalogue et mesure seulement 1,3 mégaoctets. À chaque fois qu'une nouvelle entrée est traitée, ce fichier est ouvert, et le script vérifie que le nouvel identifiant ne s'y trouve pas. Si l'identifiant ne s'y trouve pas, alors le `name` et le `trait` de cette entrée sont traités et alignés avec un identifiant *Wikidata*. À la fin de ce processus, l'identifiant `TEI` de l'entrée est ajouté au fichier de log. Cette simple modification permet donc de vérifier si une entrée de catalogue a déjà été traitée à l'aide d'un fichier dix fois moins lourd que dans la tentative précédente.

Ce processus permet de vérifier quelles entrées ont déjà été traitées, mais il ne permet pas de savoir si une personne se retrouve d'une entrée sur l'autre. Si une telle récurrence est identifiée, il serait possible de récupérer les résultats déjà disponibles, plutôt que de réaliser à nouveau l'alignement avec *Wikidata*. Non seulement cela pourrait rendre le script plus rapide – le lancement des requêtes pour une entrée prenant environ 2 à 5 secondes –, mais en plus, cela éviterait de surcharger l'`API` de requêtes, ce qui, dans un cas extrême, peut faire saturer une application en ligne et perturber son fonctionnement. La difficulté consiste alors à désambiguïser les informations présentes dans le `name` (c'est-à-dire, à repérer toutes les occurrences d'une même personne dans les `name`). Cette désambiguïsation est compliquée (l'un des intérêts de la reconnaissance d'entités nommées est justement d'identifier les multiples occurrences d'une même personne). Plutôt que de trouver un moyen de désambiguïser réellement les entrées, une solution plus simple a été mise en place : chaque chaîne de caractère recherchée est enregistrée dans un fichier de log, avec l'identifiant *Wikidata* récupéré par recherche. L'intégralité du processus d'extraction d'information des `name` et `trait` est donc menée sur toutes les entrées ; toutes les chaînes de caractères à rechercher sont également construites ; cependant, une recherche n'est faite sur l'`API` que si elle n'a jamais été faite auparant. Du fait de la taille du jeu de données, un problème apparaît cependant : plusieurs chaînes de caractères étant construites pour chaque entrée de catalogue, l'utilisation d'un seul fichier de log pour contenir toutes ces recherches amènerait à la création d'un immense fichier (plusieurs centaines de milliers de lignes). Il serait alors nécessaire de le parcourir intégralement à chaque fois qu'une requête est lancée ; le script optimisé risquerait alors d'être moins efficace qu'avant son optimisation. Le choix a donc été fait de créer automatiquement et d'utiliser plusieurs fichiers de log : les chaînes de caractères y sont enregistrées en fonction du premier caractère qu'elles contiennent, avec un fichier de log par lettre. Ainsi, avant de lancer une recherche, le fichier à parcourir est bien plus court (2060 entrées pour la lettre « e »).

Avec toutes ces optimisations, le script a pu fonctionner en utilisant au maximum 3 gigaoctets, soit l'équivalent de naviguer sur internet en ayant un logiciel de traitement de

texte ouvert (sur ma machine). Cependant, l'utilisation fichiers de log faite ici – qui a elle même été optimisée – demande de faire un arbitrage. En n'enregistrant pas de données et en lançant des requêtes à chaque fois, c'est la charge processeur qui augmente, ainsi que la consommation de courant due à internet. En enregistrant les requêtes déjà lancées, ces charges là sont diminuées, mais l'utilisation de mémoire vive augmente (il faut charger en mémoire et lire de larges fichiers). De ces deux deux différentes approches (augmenter la charge processeur ou augmenter la mémoire vive) il est nécessaire de déterminer laquelle est la plus efficace. C'est un des objectifs des tests menés sur l'algorithme d'alignement avec des identifiants *Wikidata*, comme nous allons le voir maintenant.

4.3.5 Évaluer du script : performance, qualité des données extraites de *Wikidata* et comparaison avec d'autres projets

Le processus d'alignement avec des entités *Wikidata* n'est pas parfait, et implique nécessairement un taux d'erreur. Au vu de la taille du corpus, la correction des résultats est un processus particulièrement chronophage. Pour pouvoir utiliser les données produites avant cette correction, il est nécessaire de trouver une manière d'évaluer la qualité des résultats. Des tests ont été réalisés pour faire cette évaluation. Ils ont également été utilisés durant tout le développement de ce processus, afin de comparer les différents algorithmes d'extraction et de requêtes entre eux. Les tests ont également permis de mesurer l'impact de chaque paramètre de recherche dans l'obtention d'un résultat valide. Enfin, l'évaluation des algorithmes permet de comparer sa version optimisée et non-optimisée.

Comme cela a déjà été dit, les tests ont été lancés sur un jeu de données représentatif contenant 200 `name` et `trait`, choisis dans différents catalogues ; ce jeu de test représentant également les différents types de noms possibles (de personnes nobles, de lieux...) ; enfin, la proportion de `trait` dans le jeu de données complet et dans celui de test est la même. Deux catégories de tests ont été réalisés. Le premier, décrit plus haut, concerne l'impact de chaque paramètre dans l'obtention d'une entité correcte sur *Wikidata* (B.1). Le second sert à mieux identifier les performances de l'algorithme final (B.2).

Ce dernier test réalise l'intégralité du processus d'extraction d'informations et d'alignement avec *Wikidata* sur l'ensemble du jeu de test. Ce processus est réalisé trois fois sans utiliser des fichiers de log et trois fois en les utilisant (c'est-à-dire, en inscrivant les recherches déjà faites dans un fichier et en vérifiant avant chaque recherche si une chaîne de caractère a déjà été recherchée). Faire plusieurs fois de suite le même processus permet de les variations de performance induites par l'utilisation de fichiers de log. La performance d'un script, en terme d'utilisation des processeurs et de la mémoire vive, est assez difficile à quantifier ; elle peut également varier d'une machine à une autre ; enfin, il n'est pas possible de mesurer la charge supplémentaire infligée à un serveur distant lors de l'utilisation répétée de sources de données en ligne. Le seul critère retenu a donc été le

temps d'exécution. Le script étant lancé trois fois en utilisant des fichiers de log et trois fois sans, les deux temps d'exécution $\bar{t}$ correspondent au temps moyen nécessaire à traiter les 200 entrées du jeu de test. Soit t_i le temps pris pour traiter une fois le jeu de test, la moyenne $\bar{t}$ correspond à :

$$\bar{t} = \frac{1}{3}\sum_{i=1}^{3} t_i$$

La difficulté pour mesurer les temps d'exécution t_i en utilisant des fichiers de log est que, en dehors des tests, ces fichiers sont de plus en plus volumineux ; le temps utilisé pour les parcourir augmente en conséquence. Pour que ces t_i soient représentatifs, de faux fichiers de log sont créés avant le lancement des tests : ceux-ci comprennent des fausses données qui correspondent environ aux résultats obtenus pour 30000 recherches lancées sur l'`API` (soit un peu moins que les 82000 entrées du jeu complet, en prenant en compte que certaines recherches ne retournent pas de résultat, ce qui allège d'autant le fichier à parcourir). Le temps d'exécution $\bar{t}$ en utilisant des fichiers de log est de 88,3 secondes ; sans ces fichiers, $\bar{t}$ est de 92,5 secondes. Si la différence entre les deux résultats n'est pas immense, l'utilisation de fichiers de log permet également la charge processeur et l'interaction avec *Wikidata*, ce permet d'autres améliorations au delà du seul temps de performance.

À part cette seule évaluation de performance, les tests sur l'algorithme final (B.2) servent à mesurer la qualité des résultats obtenus. La mesure de performance pour un processus d'alignement est le score F1, qui prend à la fois en compte la précision (le nombre de vrais positifs obtenus parmi tous les résultats obtenus) et le rappel (le nombre de vrais positifs obtenus parmi tous les vrais positifs). Celui-ci est de 67,4%, ce qui marque une nette amélioration (près de 20 points) de l'utilisation seule du prénom et du nom de famille usuel extrait du `name`.

Ce score est également intéressant en comparaison avec d'autres projets de Named Entity Linking (NIL) , sous-domaine du traitement automatisé du langage qui vise à associer une entité avec une base de connaissances externes. Aicha Soudani et al. ont, en 2018, présenté les résultats d'un projet de NIL de lieux sur un corpus relativement proche du notre : des textes littéraires du XIX$^{\text{ème}}$ s. encodés en `XML-TEI`. [29]. Les méthodes utilisées par ce projet diffèrent largement de celles présentées ici : les auteur.ice.s s'appuient fortement sur l'apprentissage machine, en identifiant d'abord des entités dans le texte à l'aide de `SEM` avant de lier les entités à différentes bases de connaissance en utilisant `REDEN`. Leur projet présente donc une complexité supplémentaire, puisqu'il demande de repérer les entités, ce qui n'est pas le cas ici (elles se trouvent toutes dans des balises `name`). Cependant, c'est cette deuxième étape qui nous intéresse ; les entités ont déjà été identifiées, ce qui nivelle cette différence entre les projets. Trois bases de données en ligne ont été uti-

29. **soudani_adaptation_2018**.

lisées : *DBPedia*, *DataBNF* et *Wikidata*. En comparaison avec ce projet, il est intéressant de remarquer que le processus d'alignement avec *Wikidata* développé par *MSS / Katabase* obtient un score F1 très satisfaisant (67,4%). Celui-ci dépasse les scores obtenus par Soudani et al. dans leur alignement avec *Wikidata* (61,1%) et *DBPedia* (53,6%) (voir le détail des résultats en annexe : B.3). Les résultats comparativement faibles de Soudani et al. s'expliquent en partie par le fait que `REDEN` recherche des candidats pour le liage d'entités en établissant une correspondance parfaite des caractères (*exact string match*) [30] [31]. Les scores comparativement positifs du projet *MSS / Katabase* sont peut être également rendus possibles par la quantités d'informations disponibles sur chaque personne dans le corpus, récupérée grâce à un travail d'extraction des données sur l'ensemble du `trait` et du `name`.

4.3.6 Conclusion : retour sur l'extraction d'informations des catalogues et sur l'algorithme d'alignement avec *Wikidata*

L'algorithme liant les noms d'auteur.ice dans les catalogues avec des entités *Wikidata* propose une réponse à plusieurs difficultés techniques propres à la reconnaissance d'entités par la détection de motifs sur un corpus de données varié. Parmi toutes ces difficultés possibles, les trois les plus importantes sont le bruit dans les données d'origine, le bruit produit par l'extraction de données et enfin la difficulté à réaliser un alignement entre des documents historiques et des bases de données contemporaines. La solution proposée s'appuie sur un fonctionnement modulaire, qui s'adapte aux données produites et prend en compte la possibilité que du bruit se soit introduit à une étape ou à une autre. De différence de résultats obtenue avec Soudani et al., il faut retenir que la technique utilisée n'est pas toujours garante de la qualité des résultats obtenus. Plutôt que de chercher à établir des correspondances exactes, l'approche décrite dans ce mémoire, basée sur une détection de motifs *low-tech* a permis de développer une méthode très précisément adaptée à la fois au corpus et à *Wikidata*. Ce qui est potentiellement perdu par la détection de motifs est donc corrigé par l'algorithme menant des requêtes sur *Wikidata*.

L'alignement avec des entités *Wikidata* est la meilleure manière de corriger le bruit dans les données du projet : en liant les différents `name` à des identifiants *Wikidata*, il devient possible d'extraire des données structurées d'une source externe, pour construire une base de connaissance plus fiable : même si un corpus semi-structuré facilite le travail d'extraction d'informations, l'accès aux informations reste relativement complexe ; il existe de plus des biais dans les informations contenue par le corpus (ce qui est mis en avant

30. **soudani_adaptation_2018**.

31. Il est également à remarquer que Soudani et al. disposent d'un score d'« exactitude totale du liage » (*Overall linking accuracy*) qui est nettement plus élevé (entre 70 et 85%) ; la méthode de calcul de ce score, qui mesure la performance de l'ensemble du processus de NIL, n'est cependant pas décrite dans l'article. Il est donc impossible d'y comparer les résultats de *MSS / Katabase*.

dans les `trait` est avant tout ce qui est susceptible d'intéresser le public de l'époque). En se servant de ce liage d'entités pour constituer une base de données, il est possible de chercher à corriger ce biais dans les données, tout en rassemblant assez d'informations pour pouvoir faire une analyse des facteurs déterminant le prix d'un manuscrit sur le marché du XIX^{ème} s.. C'est la manière dont est constitué le jeu de données issu de *Wikidata* qui est expliqué ci-dessous.

4.4 Après l'alignement, l'enrichissement : utiliser SPARQL pour produire des données structurées

Le liage des noms contenus dans le `name` avec *Wikidata* n'est qu'une étape, et non la fin en soit du processus – bien que ce soit la partie la plus difficile. Les identifiants ainsi récupérés servent de « porte d'entrée » aux données disponibles sur la base de connaissance : une fois que ces identifiants sont connus, il est possible de récupérer automatiquement via `SPARQL` des données sur les auteur.ice.s et autres entités présentes dans les catalogues. Obtenir ces informations permet avant tout de mieux situer l'auteur.ice d'un manuscrit, afin mieux le statut d'une personne et de comprendre quelles informations biographiques influencent le prix d'un manuscrit. Mais le liage d'entités nommées peut avoir d'autres fins et permettre d'enrichir le corpus de catalogues de plusieurs manières, comme nous le verrons.

4.4.1 Comprendre les particularités des modèles sémantiques de données

Un bref retour sur la manière dont s'organisent les données contenues dans *Wikidata* permet de mieux comprendre le problème que peut poser la diversité du corpus pour la constitution d'une base de connaissances propre au projet. *Wikidata*, comme de nombreuses autres bases en lignes (*DataBNF, DBPedia...*), permet d'accéder à des informations stockées dans une base de données en graphe. Ce type de base de données a un fonctionnement très particulier qui influence la constitution d'une base de connaissance à partir des identifiants *Wikidata* récupérés à l'étape précédente. Dans une base de données en graphe, les données sont encodées en `XML-RDF`. Ce format – dit sémantique – contient des données liées sous forme de « triplets » sujet–prédicat–objet, où :

— le sujet est la ressource principale.

— le prédicat est une propriété du sujet, qui caractérise une relation avec une autre ressource, l'objet.

— l'objet est une ressource secondaire : c'est la valeur d'un prédicat.

FIGURE 4.11 – Exemple de relation sujet – prédicat – object

Le principe des triplets RDF est mieux exprimé sous forme graphique (4.11) :
Trois particularités supplémentaires définissent les formats sémantiques :

— Toutes les « ressources » peuvent être tour à tour sujet ou objet. L'exemple du dessus, par exemple, aurait pu être réécrit sous la forme : « Les porteuses » a été peint par Natalia Gontcharova. Dans ce cas, Natalia Gontcharova est l'objet et *Les porteuses* est le sujet. Par conséquent, une base de données RDF est une base de données en graphe ; elle peut être représentée sous la forme d'un réseau de ressources qui entretiennent des relations bilatérales entre elles. Il n'y a pas de hiérarchie entre les informations, contrairement à une base de données XML classique.

— L'ensemble des ressources et des prédicats d'une base de données en graphe sont définis et disposent d'un identifiant unique (certains de ces identifiants ont été récupérés grâce au liage d'entités nommées).

— Si le langage SPARQL offre une syntaxe commune, chaque base de connaissance peut utiliser des vocabulaires particuliers qui sont organisés en « ontologies ». Une ontologie correspond à la définition d'un ensemble de catégories, de propriétés et de relations qui unissent des données et des concepts ; cet ensemble est complété par une modélisation (souvent sous forme graphique), qui indique la relation entre les différents termes de l'ontologie [32]. Ceux-ci sont souvent liés de façon hiérarchique (plusieurs termes spécifiques pouvant être dérivés d'un terme générique). Au sein d'une ontologie, chaque terme a lui-même un identifiant unique, ce qui garantit une implémentation uniforme pour l'ontologie. Une même base de connaissance peut utiliser plusieurs ontologies. Celles-ci sont définies à l'aide d'espaces de noms [33], c'est-à-dire par des identifiants qui permettent de différencier les ontologies. Les prédicats, plus particulièrement, sont définis selon une ontologie particulière.

SPARQL a l'avantage de permettre de récupérer des données propres sur des bases en ligne et offre une syntaxe unique partout où il est implémenté. Cependant, le troisième point complexifie son utilisation, ainsi que la définition des données à récupérer : les prédicats sont décrits avec une grande précision et selon des vocabulaires spécifiques ; par conséquent, une information analogue peut être représentée par différents prédicats selon le type de donnée qui est requêtée (une personne, une sculpture...). Dans l'ontologie *Wikidata*, la création d'un texte et la création d'une peinture ne correspondent pas au

32. **noauthor_ontology_2022**.
33. **noauthor_namespace_2022**.

même prédicat. Pour que les données soient utilisables, il faut être très spécifique quant aux informations recherchées.

4.4.2 Quelles données rechercher via SPARQL ?

La question des données qui doivent être récupérées, et donc de la base de connaissance à constituer pour le projet *MSS / Katabase* n'est pas anodine. D'abord, l'utilisation de sources externes peut être utiliser pour corriger des biais dans les données originelles (où ce sont les auteur.ice, éditeurs et éditrices de catalogues qui décident des informations à intégrer). Ensuite, si l'objectif premier est de récupérer des données pour mener une étude économétrique, le liage avec *Wikidata* peut également permettre d'autres enrichissements et fonctionnalités. Enfin, cette étape représente une difficulté technique : le corpus de catalogues et les entités récupérées sont très diverses, tandis que SPARQL utilise au contraire des vocabulaires très spécifiques. Il faut donc construire des requêtes très étendues, afin d'obtenir des résultats pour tous les types de données. Les 18899 entités avec lesquelles les entrées de manuscrits ont été alignées peuvent se classer en de nombreuses catégories. Sur *Wikidata*, une entité est une « instance » d'une classe plus large. C'est en fonction de ces classes qu'il faut construire les requêtes : les propriétés recherchées pour chaque entité peuvent être spécifiques à une classe, mais pas à l'entité elle-même. En suivant la classification de *Wikidata*, les entités présentes dans le corpus appartiennent aux catégories suivantes :

— personnes humaines ; cette catégorie est la plus fréquente (12090 occurrences)
— noms de familles (3180 entités)
— communes françaises (586 occurrences)
— peintures et sculptures (respectivement 520 et 236 entités)

Cette variété – présentée sous forme graphique en annexes (C.1) – peut s'expliquer en partie par le taux d'erreur dans l'alignement avec *Wikidata* : il est par exemple probable que les peintures et sculptures soient sur-représentées parmi les entités *Wikidata* liées aux catalogues. Cependant, toutes ces « erreurs » ne correspondent pas forcément à des résultats qui ne sont pas pertinents. Par exemple, l'algorithme peut aligner un.e écrivain.e avec un de ses ouvrages, ou une personne avec son portrait. Des résultats erronés peuvent toujours garder une forme de pertinence. Il est d'autant plus important de construire des requêtes SPARQL qui se concentrent pas uniquement sur des personnes. Cependant, il n'est pas possible de s'adapter à l'intégralité de la diversité du corpus. Le choix a donc été fait de se concentrer sur les catégories les plus pertinentes : les personnes, les familles, et les œuvres artistiques et littéraires. Non seulement ces catégories contiennent la grande majorité du corpus (16026 entités, soit 84,79%), mais ces catégories sont les plus à même de contenir des entités pertinentes. Il a été choisi de ne pas faire de requête spécifique sur

les lieux, puisque *Wikidata* présente peu d'informations pour les entités de la catégorie « communes françaises ».

Un nombre assez conséquent de données ont donc été requêtées avec `SPARQL`, du fait des spécificités des bases de données en graphes, de la variété des entités *Wikidata* auxquelles les manuscrits sont liées, et enfin du fait de la variété du corpus lui même. Ces informations récupérées correspondent aux différentes catégories de *Wikidata*.

— Pour les personnes et les familles, les informations suivantes sont récupérées sur *Wikidata* :

  — Le genre de la personne ;

  — Sa nationalité, afin de voir si l'origine d'une personne influence le prix d'un manuscrit ;

  — Les langues parlées par une personne ; là encore, l'objectif est d'étudier l'impact de l'origine d'un.e auteur.ice sur un prix.

  — Les date de naissance et de décès, afin de placer un manuscrit dans une époque et de voir comment son ancienneté ou sa contemporanéité en influencent le prix.

  — Le lieu où une personne est née, où elle a vécu et où elle est morte, pour des raisons analogues.

  — La manière dont la personne est morte. Si cette information peut sembler anecdotique à un public contemporain, les catalogues de ventes sont marqués par un goût du sensationnel, et la manière dont une personne est morte est souvent mentionnée, notamment en cas d'exécutions.

  — La religion d'une personne : il peut être intéressant d'étudier si, et comment, ce critère influence l'évolution d'un prix.

  — Les titres de noblesse d'une personne.

  — L'éducation qu'a reçu une personne, afin de mieux situer ses occupations et d'analyser l'impact du niveau et du type d'éducation sur le prix.

  — L'occupation d'une personne, et les fonctions précises qu'elle a occupées : là encore, il est intéressant de situer l'impact de la carrière sur le prix et de voir quelles occupations sont corrélées avec des prix élevés sur le marché des manuscrits.

  — Les prix et distinctions reçus par une personne. À l'aide de ce critère, il est alors possible de chercher à répondre à cette question : la célébrité d'une personne de son vivant impacte-elle le prix de se manuscrits ?

  — Les organisations et institutions dont la personne est membre (Académie française, Franc-maçonnerie...)

  - Le nombre d'œuvres écrites ou réalisées par une personne. Là encore, c'est une tentative de mesurer l'impact ou la célébrité d'une personne : les manuscrits de quelqu'un ayant beaucoup écrit sont ils plus chers que les manuscrits d'une personne ayant peu écrit ?
  - Le nombre de conflits auxquels une personne a participé. Ce critère de recherche permet de quantifier l'importance d'un personnage militaire.
  - Des images, telles que le portrait et la signature.
- Pour les créations littéraires, ce sont des informations bibliographiques qui sont avant tout récupérées ; pour les autres œuvres d'art, des informations analogues sur le contexte de création sont retenues.
  - Le titre de l'œuvre.
  - Son auteur.ice, pour étudier si certain.e.s auteur.ice.s sont susceptibles d'influencer le prix d'un manuscrit.
  - La date de création de l'œuvre, afin de savoir si l'époque d'origine influence le prix. Pour les livres, la date publication est également récupérée.
  - La requête récupère aussi la maison d'édition d'un livre.
  - Les dimensions et matériaux d'une œuvre d'art sont également d'intérêt.
  - Enfin, le genre et le mouvement dans lequel s'inscrivent une œuvre sont d'intérêt : ces informations pourraient permettre de voir si une hiérarchie des du goût influence le prix d'un manuscrit.
- Pour finir, afin de pouvoir éventuellement enrichir nos données avec d'autres sources externes à *Wikidata*, des identifiants uniques ont été récupérés afin de donner accès à d'autres bases de données en ligne : les identifiants VIAF (Fichier d'autorité international virtuel), ISNI (International Standard Name Identifier), de la Bibliothèque nationale de France, de la Bibliothèque du Congrès américain, ainsi que les identifiants IDRef. Certaines insitutions, comme la BnF, rendent leurs données accessibles via `SPARQL` ; la récupération de ces identifiants faciliterait grandement les enrichissements ultérieurs depuis d'autres sources de données.

Comme on l'a dit, l'objectif principal de l'alignement avec *Wikidata* est de produire des données pour calculer des régressions linéaires, ce qui permettrait d'étudier les déterminants du prix d'un manuscrit sur le marché du XIX$^{\text{ème}}$ s.. Cette récupération d'informations en masse ouvre d'autres possibilités. Entre autres, de nombreuses données géographiques ont sont récupérées (lieu de naissance, de décès, d'inhumation et de résidence). Il est ensuite possible de récupérer les coordonnées de ces lieux, afin de construire une cartographie des auteur.ice.s dont les manuscrits circulent sur le marché du XIX$^{\text{ème}}$ s. parisien. C'est d'ailleurs souvent à des visées cartographiques et de géoréférencement que

sont menées des campagnes de reconnaissance d'entités nommées ; dans les dernières années en France, de nombreuses études ont été menées pour développer des cartographies à partir de textes littéraires français encodés en TEI (Soudani et al. [34], [35]). En développant une approche cartographique, le projet *MSS / Katabase* pourrait apporter de nouvelles possibilités pour de telles études : le corpus de catalogues étant une source secondaire sur l'histoire littéraire française, une approche géographique permettrait d'étudier les origines géographiques des auteur.ice.s, plutôt que d'analyser les lieux représentés dans leurs œuvres. Cette possibilité n'est pas anodine, puisqu'elle permettrait de mettre en relation la « parisianité » avec la construction du canon littéraire à Paris. Il serait également possible d'étudier la circulation des productions culturelles, et leur rayon d'influence. En croisant les données géoréférencées avec des informations chronologiques (dates de naissance et de mort...), ces questions peuvent également être étudiées de façon historique : comment l'influence de l'origine géographique sur la réception d'une œuvre évolue au fil des siècles ? Répondre à ces questions n'a pas été possible dans le cadre de mon stage ; cependant, grâce à l'enrichissement de données via SPARQL, il de telles études deviennent possibles, et les données pour mener ces analyses sont au moins en partie déjà disponibles. Produire des informations normalisées et exploitables pour la recherche implique donc de produire des données qui puissent être réutilisées avec d'autres problématiques de recherches.

4.4.3 Présentation générale

L'algorithme de récupération des informations sur *Wikidata* est considérablement plus simple que ce qui a été présenté lors de l'extraction d'informations des catalogues et de l'alignement avec des entités *Wikidata* : lors de ces étapes, les principales difficultés résultaient du bruit dans les données et de la nature « semi-structurée » des entrées de catalogues, qui demandait de s'adapter à de nombreux cas de figure. Une fois que des identifiants *Wikidata* ont été extraits, le processus est bien plus simple – comme cela apparaît dans la figure 4.12. Un identifiant étant unique et servant de clé d'entrée à une base de données en graphe très structurée, un comportement uniforme peut être adopté pour récupérer des données issues de *Wikidata*. Il ne s'agit plus que, pour chaque identifiant *Wikidata* récupéré, de lancer des requêtes SPARQL et d'en stocker le résultat dans un fichier. Le comportement de l'algorithme est donc toujours le même.

L'algorithme commence par vérifier si un identifiant a déjà été traité dans un fichier de log ; l'utilisation de ces fichiers évite d'avoir à recommencer la récupération d'informations de *Wikidata* à chaque interruption du script, ce qui est essentiel puisque cette étape dure plus de dix heures. Si l'identifiant n'a pas été traité, alors plusieurs requêtes SPARQL sont lancées sur *Wikidata*. Les résultats sont retournés en JSON ou XMLdans des formats

34. **soudani_adaptation_2018**.
35. **frontini_annotation_2016**.

définis dans une spécification du W3C [36]. Ceux-ci ont un but descriptif (ils décrivent la requête et les données retournées) ; les documents SPARQL sont donc très complets et peu malléables ; c'est pourquoi ils sont transformés en une base de connaissance au format JSON. Celle-ci, comme nous le verrons, ne contient que les données obtenues dans un format plus malléable.

Les requêtes SPARQL sont faites sur un serveur distant via le protocole HTTP et demandent parfois au serveur de traiter de très grandes quantités de données. Des erreurs peuvent donc avoir lieu. Pour permettre au script de fonctionner malgré ces erreurs, un système de gestion des erreurs a donc été mis en place. Par défaut, il est demandé au serveur de *Wikidata* de fournir les résultats de la requête en JSON, format très léger et malléable. Deux erreurs peuvent alors arriver : soit le JSON retourné par le serveur est mal formé (et ne peut donc être traité), soit la requête met plus excède la durée maximale autorisée (qui est d'une minute). Dans ce cas, la requête est lancée une seconde fois au serveur, mais cette fois-ci le format de réponse demandé est le XML ; cela permet d'éviter que le document soit mal formé ; il est également possible que la requête s'exécute dans le temps imparti la seconde fois. Si une erreur a encore lieu à la seconde requête, alors une entrée vide est associée à l'identifiant *Wikidata* dans la base de connaissance, pour signifier qu'aucune réponse n'a pue être obtenue.

36. **beckett_sparql_2013**.

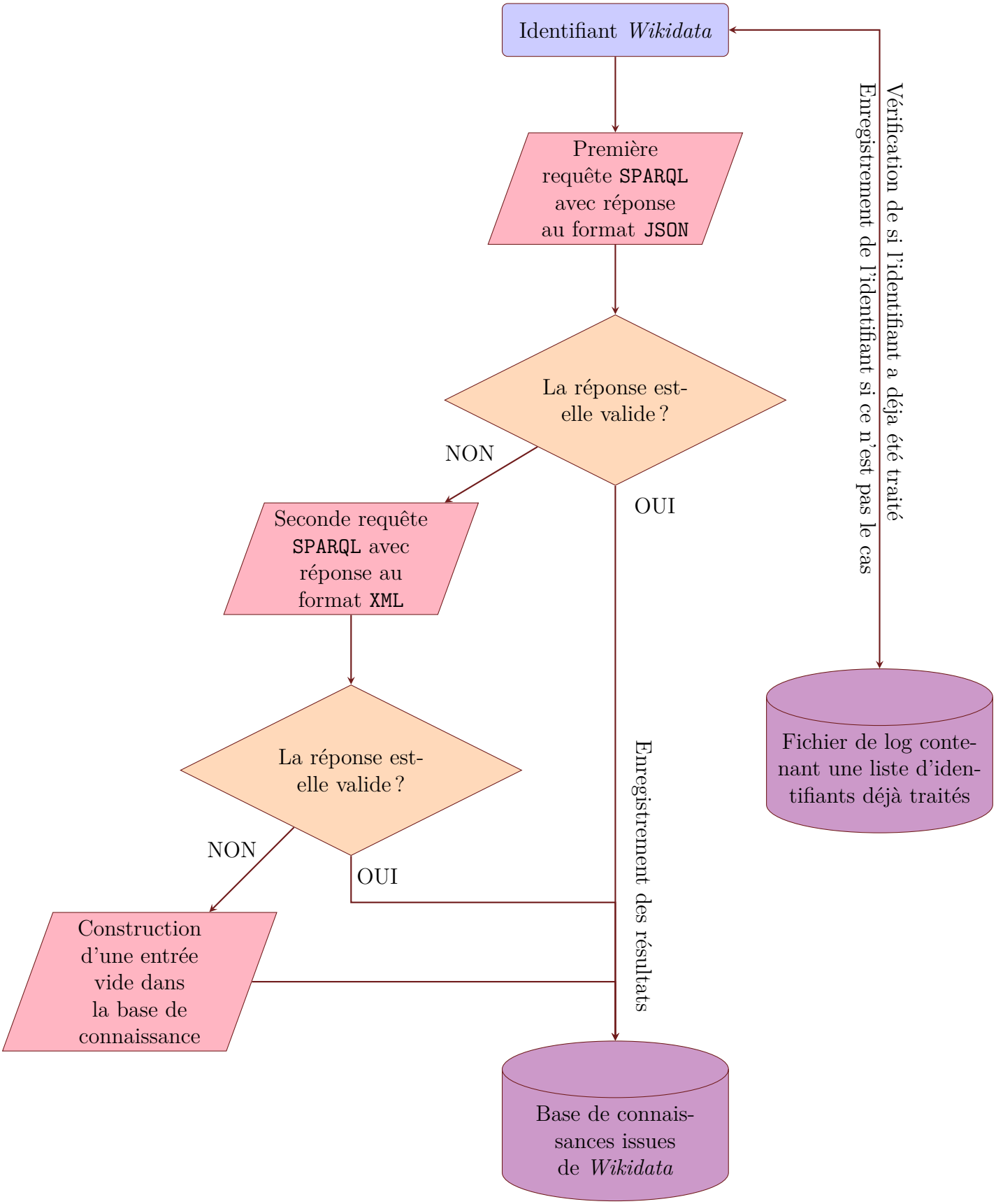


FIGURE 4.12 – L'algorithme de constitution d'une base de connaissances issues de *Wikidata* à l'aide de SPARQL

```
SELECT ?painting ?paintingLabel
WHERE {
    ?painting wdt:P170 wd:Q232391 .

    SERVICE wikibase:label {
        bd:serviceParam wikibase:language "en" .
    }
}
```

Code source 4.18 – bla

4.4.4 Développer un comportement uniforme pour produire des données exploitables à partir un corpus hétérogène

D'un point de vue technique, SPARQL suit la logique des « triplets » présentée ci-dessus. Le langage permet d'interagir avec une base de données en graphe au format RDF (récupérer des données, en envoyer, en mettre à jour ou en supprimer). Récupérer des données via SPARQL revient à demander au langage d'extraire de la base de données l'ensemble des valeurs qui correspondent à un triplet sujet-prédicat-objet. Par exemple, récupérer tous les tableaux peints par Natalia Gontcharova est possible à l'aide de la requête :

ALORS ALORS DE QUOI CA PARLE - le fonctionnement de sparql en triplets, en reprenant l'exemple plus haut - la division en plus petites requêtes pour éviter les timeouts - les result formats définis par W3C (avec des exemples, en xml et json, en annexes), et donc vite fait le processus de conversion - la construction d'une base de données en sortie : format (json), structure. Cette base de connaissance abandonne l'aspect auto-descriptif des réponses SPARQL : en plus de contenir les données renvoyées par SPARQL, les formats de réponse définis par le W3C [37] décrivent la requête (en stockant un ensemble de variables) et les données retournées (en définissant systématiquement un type de données, et en associant à ce type une *Uniform Resource Identifier* (URI)). Le deuxième, et le plus important problème des formats de réponse SPARQL, est qu'ils contiennent de nombreuses redondances. Comme cela a été dit, une requête SPARQL associe une variable à une « question » ; celle-ci permet d'associer une ou plusieurs valeurs à cette variable. Une requête comprenant plusieurs variables peut donc retourner un nombre de valeurs différent pour chacune de ces variables. SPARQL ne peut alors pas déterminer de lien logique entre les valeurs obtenues pour ces variables. SPARQL fait donc le produit cartésien entre toutes la valeurs possibles pour chaque variable : il y a alors autant de « solutions à la requête » [38] qu'il y a de combinaisons possibles de résultats. Par exemple, si trois

37. **beckett_sparql_2013**.
38. **beckett_sparql_2013**.

variables sont requêtées et que SPARQL retourne trois résultats pour chaque variable, alors le nombre de « solutions à la requête » est de $3 \times 3 \times 3 = 27$.

La réponse retournée comprend alors plusieurs

Ici est détaillée

— la requête SPARQL lancée (subdivisée en plusieurs petites requêtes).

— le format de sortie produit à partir des données renvoyées par SPARQL

4.4.5 Minimiser la perte : optimisation et gestion des erreurs

Comme des quantités massives de requêtes sont lancées, et que de très nombreuses informations sont demandées, des erreurs peuvent avoir lieu, et notamment des erreurs de *timeout* (le temps d'exécution dépasse la durée autorisée). La gestion de ces erreurs est décrite ici.

4.4.6 Lier la TEI aux données nouvellement produites

Cette courte section détaille la mise à jour des fichiers TEI avec les identifiants *Wikidata*, ce qui permet de faire le lien entre les entrées de catalogues et les données issues de *Wikidata*.

4.5 Des données à la monnaie : premiers résultats de l'étude

Sous réserve que l'étude des régressions linéaires ait été fait à temps (ce qui n'est pas garanti), j'aimerais ici présentés les premiers résultats sur les facteurs de l'évolution des prix.

Troisième partie

Après la TEI : l'application web *Katabase*, interface de diffusion des données

Chapitre 5

Design d'interfaces dans un projet d'humanités numériques : l'application web *Katabase*

Ce chapitre s'intéresse aux relations entre *web design*, données textuelles et humanités numériques, à partir de l'exemple du site web développé pour le projet *Katabase*.

5.1 Le design d'interfaces : une reconfiguration des méthodes de recherche et une transformation du corpus

Cette section s'intéresse aux nouveautés apportées par le design d'interfaces dans les humanités numériques. On s'intéresse à la manière dont le design d'interfaces (et le design de façon générale) transforme les méthodes de recherche « habituelles », mais aussi une transformation du rapport aux documents.

5.1.1 Le design comme inversion des méthodes

Avec les humanités numériques, les questions de design et de structuration deviennent centrales, depuis la conception de schémas `TEI` (qui demandent de mettre en forme un document pré-existant) et d'ontologies jusqu'au développement d'interfaces et de sites web. Parmi ces questions « formelles », le design d'interfaces occupe cependant une place particulière. En effet, dans la plupart des aspects des humanités numériques, le rapport entre questions techniques et scientifiques est clairement établi ; la question scientifique préexiste, et la technique sert surtout à répondre à cette question (comme cela a été le cas jusqu'à dans la « pipeline » jusqu'ici). Cette hiérarchie entre théorie et

pratique reste somme toute assez traditionnelle et correspond aux méthodes scientifiques établies.

Avec le design d'interfaces, ce rapport établi s'inverse. En effet, le design ne cherche pas à répondre à une question. Tout au plus, il répond à un cahier des charges (il faut, à minima, permettre de diffuser des données de façon lisible par des êtres humains). C'est avec la pratique du design que naissent les problématiques, parmi lesquelles :

— Comment organiser les différentes parties d'une page pour que celle ci soit lisible ?

— Comment organiser la relation entre les pages pour qu'un site web soit facilement navigable ?

— De quelle manière l'apparence d'un site détermine la réception des contenus ?

— En quoi le design d'un site web construit ou bouscule des habitudes et des formes d'utilisation chez ses utilisateur.ice ?

Toutes les questions posées par le design n'attendent pas nécessairement de réponse. Cependant, force est de constater que ce domaine appelle à une nouvelle approche pour des chercheur.euse.s et ingénieur.e.s issu.e.s des humanités ; ces questions visuelles amènent à une approche semblable à celle de la recherche-création et demandent de développer un nouveau rapport à la technique.

5.1.2 Interface et document

En plus de perturber nos méthodes, la conception d'interfaces influence la perception des documents. Dans le cas du projet *Katabase*, le site web opère une médiation, il implique de une « scénographie » autour des catalogues de vente. Ceux-ci et les manuscrits qui y ont décrits sont intégrés à des pages, inclus dans un parcours, accessibles depuis différents points d'entrée. En plus de cette scénographie, les catalogues sont littéralement traduits, depuis la `TEI` vers le format `HTML`. Là où la `TEI` est un format de balisage sémantique (c'est la signification des éléments guide l'encodage), le `HTML` est pensé pour un balisage formel (le texte est balisé en fonction de la forme que l'on souhaite obtenir). Cela implique une perte d'information (les métadonnées du `teiHeader`) et l'éloignement d'une approche philologique du texte. Enfin, le site internet marque un éloignement intellectuel avec les documents : le catalogue n'y est plus l'unité intellectuelle dominante, alors qu'il restait l'un des critères structurants des fichiers `TEI` (un fichier représentant un catalogue). Sur le site web, on peut accéder directement aux éléments vendus, sans avoir à passer par les catalogues. Dans le contexte d'un projet issu de la littérature, toutes ces opérations ne sont pas neutres et méritent d'être explicitées. Pour mieux identifier ce que ces transformations impliquent, il peut être intéressant de revenir à la « roue » de Sahle [1].

1. **sahle_digital_2016**.

5.2 La conception d'interfaces, un problème pour les humanités numériques ?

Cette section s'intéresse aux rôle des interfaces en humanités numériques.

5.2.1 Pour une approche pragmatique du design d'interfaces dans un contexte d'humanités numériques

Le design graphique demande des compétences spécifiques qui ne font pas directement partie des cursus d'humanités numériques. Il ne sert pas seulement à faire des sites qui soient « beaux », il joue un rôle essentiel en encadrant la réception des contenus présentés. Cependant, les approches plus « élaborées » de design d'interfaces demandent des financements et des techniques qui sont souvent hors de portée d'un projet universitaire. Des approches plus « critiques » du design ont également été développées dans les humanités numériques [2]. Ces approches ont tendance à être difficiles à mettre en œuvre ; leur portée critique peut aller à l'encontre de l'utilité des interfaces, en faisant de l'interface l'objet principal d'intérêt, aux dépends des contenus présentés.

À l'opposé de ces approches, ce qui est défendu dans le cadre du projet *MSS / Katabase* est une approche à la fois informée et pragmatique du *web design*. Informée, car être conscient des enjeux du design permet un meilleur positionnement en tant qu'ingénieur.e, et donc une présentation des contenus plus intéressante. Pragmatique, parce que les solutions qui sont présentées sont des solutions techniquement réalisables dans le cadre d'un projet universitaire. C'est ici qu'est présentée la charte graphique développée pour l'application web *Katabase*.

5.2.2 Rejeter les interfaces ?

Après avoir parlé de l'intérêt des interfaces et présenté l'approche suivie au sein du projet *MSS / Katabase*, cette partie s'attache à développer une critique des interfaces. À partir d'une approche historique des interfaces graphiques, des contextes dans lesquelles elles se sont développées, nous revenons sur les concepts centraux à leur développement que sont la notion d'utilisateur et de design d'expérience. Il ne s'agit pas de remettre en cause l'utilisation d'interfaces, mais de défendre une approche critique et consciente de l'impact que la standardisation des « expériences utilisateur » sur internet peuvent avoir sur la diffusion des connaissances.

2. **drucker_visualisation_2020**.

Chapitre 6

Donner à voir un corpus textuel

Ce chapitre s'intéresse aux visualisations développées pour l'application web *Katabase*.

6.1 Visualisation, design et sciences : des relations complexes

Ici, on s'intéresse à la place qu'occupe la visualisation de données dans la recherche scientifique. Le rapport entre les sciences et la visualisation est loin d'être simple et unidirectionnel : cette dernière n'est pas juste un outil, une méthode utilisée dans la recherche scientifique pour des raisons pratiques. Il est plus intéressant de penser la visualisation (et donc le design) et les sciences comme des domaines en interaction, qui s'influencent mutuellement. De la même manière que l'écriture implique des manières de penser particulières [1] en donnant au discours une existence spatiale (le texte est répandu sur une page et des renvois peuvent être fait d'un endroit de la page à un autre), la visualisation implique ses propres manières de penser et influence donc la recherche. Dans notre cas par exemple, produire des visualisations implique de s'intéresser à des informations quantifiables ; cela encourage donc une approche statistique du corpus. À l'inverse, la recherche scientifique ne fait pas qu'« utiliser » le design. Certaines pratiques sont favorisées et deviennent force d'autorité dans des disciplines scientifiques. Se créent alors des « cultures visuelles » [2] propres à ces disciplines, avec leurs traditions et motifs.

1. **masure_design_2017**.
2. **hentschel_visual_2014**.

6.1.1 L'utilisation de supports visuels dans les sciences : une longue histoire

Ici, on retrace une histoire de l'utilisation du visuel dans les sciences (au sens large : sciences « dures » et sciences humaines), à partir (entres-autres) du travail de Anne-Lyse Renon [3] et de K. Hentschel [4].

6.1.2 Une vision objective ? Visualisation et prétention à l'objectivité

Cette partie fait un retour sur la manière dont le visuel et la production de graphiques ont été utilisés comme arguments d'autorité, afin de montrer des faits de façon « objective » [5].

6.1.3 La tendance visuelle des humanités numériques

Pour finir, on fait un bref retour sur la manière dont les humanités numériques « intensifient » la tendance à la visualisation, ou complexifient le rapport entre sciences et visualisation, pour deux raisons. En premier lieu, les humanités numériques viennent avec le développement de nouveaux outils. Ensuite, les humanités numériques marquent un retour à une approche quantitative et graphique dans les sciences humaines – approche qui trouve ses sources, entre-autres, dans l'École des Annales et sa collaboration avec le Laboratoire de graphique de Jacques Bertin [6], ainsi que dans le structuralisme, où les « structures » trouvent leur meilleure représentation sous forme graphique. Cette tendance visuelle des humanités numériques n'est pas sans poser problème, puisque les visualisations sont développées par des personnes qui n'ont pas nécessairement de formation en graphisme. Une approche pragmatique de la visualisation a tendance à primer (les graphiques servent à prouver quelque chose), aux dépens d'une approche critique (les représentations graphiques sont des interprétations, où les données sont signifiantes, mais où les formes et les méthodes de représentation importent aussi).

6.2 Interpréter le corpus de manuscrits

Ce chapitre s'intéresse à la manière dont le corpus de catalogues de vente de *Katabase* a été traduit en graphiques et à la manière dont ces représentations permettent un nouveau regard sur le corpus.

3. **renon__design__2016**.
4. **hentschel__visual__2014**.
5. **renon__design__2015**.
6. **orain__laboratoire__2021**.

6.2.1 La visualisation comme objet de connaissance

Ici, on développe une analyse du corpus de catalogues et des manuscrits qui y sont décrits à partir des visualisations produites. Par leur capacité à traduire les informations sous des formes synthétiques [7], les visualisations sont des objets de connaissance qui permettent de comprendre le corpus traité.

6.2.2 La visualisation comme interprétation

Les représentations graphiques ne font pas que montrer des phénomènes. Leur rôle est moins analytique que démonstratif : elles ne révèlent pas une information qui serait cachée dans les données, mais interprètent celles-ci conformément à une problématique de recherche [8]. Représenter un jeu de données, c'est donc le lire, l'interpréter en fonction de certaines questions scientifiques. Ce processus interprétatif est donc partiel (on ne dit pas tout ce qui est dans un jeu de données, mais seulement ce qui est pertinent dans un certain contexte) ; il est aussi influencé par les propriétés graphiques des visualisations. Cette sous-section s'intéresse donc, à partir d'exemples concrets, à la manière dont les propriétés graphiques (choix de formes et de couleurs) ainsi que le pré-traitement des données et d'autres décisions techniques (représentation des prix en francs courants ou constants) influencent la lecture et la perception du corpus.

Ainsi, en s'appuyant sur une connaissance du corpus, il est possible d'approximer une véritable compréhension « humaine » d'un texte semi structuré, en utilisant uniquement des expressions régulières. Si il n'est pas possible d'identifier le sens des différents éléments, il est possible de localiser et d'extraire les éléments signifiants. Il apparaît alors que ces éléments signifiants sont, d'une certaine manière, des formes, ou des motifs qui peuvent être détectés automatiquement ; la signification peut être identifiée en fonction du type de donnée extraite (un nombre est une date). Mais, dans un corpus semi-structuré, il surtout est possible d'inférer du « sens » d'un élément à partir de sa position dans le texte. C'est dans ce cadre qu'un encodage en `XML-TEI` des documents prend tout son intérêt : les différentes parties d'un texte sont balisées sémantiquement, ce qui est essentiel au traitement automatisé du texte : il devient possible de traiter le texte non pas dans son intégralité, mais au niveau d'une unité sémantique très précise, comme le nom donné à un manuscrit (le `name`) ou une brève description (le `trait`)

Le fait de travailler sur des corpus semi-structurés permet également de mettre en approche des méthodes alternatives, « low-tech », pour le traitement du langage et l'analyse textuelle dans des projets de recherche. Avec une bonne connaissance du corpus et des données structurées, il est possible de se baser entièrement sur des solutions techniquement « simples », telles que la détection de motifs à l'aide d'expressions régulières. Plutôt

7. **hentschel_visual_2014**.
8. **drucker_visualisation_2020**.

que de s'appuyer sur de l'apprentissage machine, et donc des solutions complexes, il est intéressant de s'appuyer sur des méthodes alternatives et de développer des algorithme plus complexes, basés sur une approche modulaire, qui sont capables de cibler très précisément les éléments d'un texte pertinent pour des problématiques de recherche précises. Ces méthodes de basse technologie, qui s'intéressent à un usage intelligent des technologies plutôt qu'à des technologies « intelligentes » forment un contre modèle qui gagnerait à être mis en avant dans les projets de recherche. L'intelligence artificielle devenue un outil central et pertinent dans les humanités numériques – à la base de la reconnaissance optique de caractères. Mais elle ne doit pas seulement être une nouveauté que les projets de recherche doivent à tout prix incorporer. C'est également une technologie extrêmement polluante [9], utilisée de façon massive par des grands groupes industriels [10] et un argument de vente pour de nombreuses start-up, alors que les applications pratiques de l'intelligence artificielle soulèvent de graves problèmes éthiques [11]. La recherche en humanités numériques, fondée sur une connaissance des techniques et des documents utilisés, devrait fournir un espace pour remettre en question de telles technologies et pour développer des solutions alternatives, où la connaissance scientifique supplante l'intelligence machine. De façon modeste, le projet *MSS / Katabase* offre une telle remise en question : en s'appuyant sur une bonne connaissance de corpus textuels semi-structurés, le projet a développé une chaîne de traitement entièrement basée sur la détection de motifs. S'appuyer sur des méthodes

9. Strubell, Ganesh et McCallum (**strubell_energy_2019**) ont étudié l'impact écologique des modèles d'apprentissage profond en traitement du langage naturel. En étudiant les ressources nécessaires à l'entraînement d'un modèle, les auteur.ice.s montrent (**strubell_energy_2019**) que la mise au point d'un modèle *Transformer* peut émettre jusqu'à 626155 livres de CO_2, soit cinq fois la consommation d'une voiture durant toute sa vie (126000 livres de CO_2) et 17 fois les émissions de CO_2 émises par une personne américaine en un an (36156 livres). Il.elle.s rappellent également que les modèles d'apprentissage machine les plus performants sont aussi les plus gourmands en ressources et qu'un modèle doit être entraîné parfois plusieurs milliers de fois pour atteindre des scores satisfaisants (**strubell_energy_2019**). Avec l'augmentation générale des performances des modèles de traitement du langage naturel, l'augmentation marginale de leur performance est de plus en plus coûteuse en ressources : l'augmentation du score BLEU d'un modèle de traduction automatique de 0.1 points de performance (pour atteindre un score de 29,7) a coûté 150000 dollars en ressources environnementales et en coût de calcul (**strubell_energy_2019**).

10. *Google*, et sa filiale *Youtube*, s'appuient depuis le milieu des années 2010 de plus en plus massivement sur l'intelligence artificielle (**covington_deep_2016**). L'algorithme de suggestion de vidéos sur *Youtube* repose entièrement sur de l'apprentissage machine(*ibid.*) : en fonction de l'historique de l'utilisateur.ice, l'algorithme « apprend » à suggérer du contenu pertinent. L'usage massif de telles technologies est particulièrement polluant : pour pouvoir suggérer des vidéos pertinentes, il faut, à chaque connexion de l'utilisateur.ice, consulter deux très grandes bases de données : dans un premier temps, l'algorithme sélectionne toutes les vidéos potentiellement pertinentes, depuis l'ensemble des vidéos disponibles sur le site ; ensuite, ces vidéos sont classées en consultant une seconde base de données et en filtrant les résultats obtenus par le comportement récent de l'utilisateur.ice (ce qu'il ou elle a recherché, combien de temps il ou elle a passé à regarder une vidéo, sa réaction à celle-ci...). Le 2 décembre 2020, *Google* a licencié la docteure Timnit Gebru, alors à la tête de l'équipe spécialisée en éthique de l'intelligence artificielle, suite à la rédaction d'un article sur les impacts écologiques de l'utilisation massive par l'entreprise de modèles de traitement automatisé du langage. Ce phénomène alarmant rappelle au passage la non-neutralité des publications scientifiques provenant ce cette entreprise, dont deux ont été citées dans le présent mémoire (**mikolov_efficient_2013** ; **covington_deep_2016**).

11. À ce sujet, voir par exemple le projet *Awful AI* de David Dao, qui liste une quantité impressionnante d'utilisations préoccupantes de telles technologies.

de basse technologie ne vaut bien sûr pas seulement comme une déclaration de bonne foi : plus une solution est techniquement simple et accessible, plus elle est réutilisable, adaptable à d'autres jeux de données et plus facilement elle peut évoluer pour incorporer de nouvelles fonctionnalités.

pollution des large language models : https ://arxiv.org/abs/1906.02243

— parler du minimal computing ??

Bibliographie

Glossaire

Open source L'*open source*, ou « code ouvert » définit à la fois un statut légal et un mouvement social. L'*open source* correspond à un logicel ou programme dont le code est librement consultable en ligne et réutilisable selon différentes licences plus ou moins permissives, comme la *GNU GPL v3.0* ou la *MIT License*. Ce statut légal a beaucoup gagné en popularité, en partie pour des raisons pragmatiques (un code ouvert est maintenu gratuitement, ce qui diminue les coûts d'entreprises) et symboliques (les GAFAM se sont beaucoup associées à ce courant en raison de l'image positive qu'il donne). Le statut est également associé à plusieurs mouvements en faveur du partage et de la réutilisation des connaissances, tel que la science ouverte. Certaines franges de l'*open source* sont plus radicales et ont une conception ouvertement politique de l'accessibilité au code, accessibilité qui permettrait aux utilisateur.ice.s de regagner en autonomie, de développer de nouveaux modèles communautaires et de reprendre le contrôle sur leurs comportements en ligne et leur consommation médiatique. Parmi les représentant.e.s les plus proéminent.e.s de ce courant se trouve Richard Stallman, créateur de la *GNU Foundation* et du système d'exploitation GNU, à la base de la majorité des systèmes d'exploitation utilisant Linux.. 60, 66

API Une interface de programmation d'application (*Application Programming Interface* en anglais) est un protocole qui pemret à un programme de communiquer avec un autre programme. Ce protocole documenté, correspond à un ensemble d'opérations permettant à un programme « consommateur » d'utiliser des fonctionnalités d'un programme « fournisseur », comme par exemple de récupérer ou d'envoyer des données au fournisseur.. 14

Base de données en graphe Une base de données en graphe sert à stocker des données reliées entre elles au sein d'un graphe ou d'un réseau. Les données correspondent aux nœuds du graphe et les relations sont représentées sous forme d'arrêtes. Souvent, données et relations disposent d'identifiants uniques définis grâce à des ontologies. L'interaction avec de telles bases de données est permise grâce à des languages de requêtes, comme `SPARQL`. La représentation en graphe a pour but de mettre l'accent sur l'interrelation entre les données et de permettre des liens complexes entre elles. [12].

12. **noauthor_graph_2022**.

71, 72, 76

Dictionnaire Un dictionnaire est un format de données structuré en python qui associe à une donnée – dite clé – une ou plusieurs données nommées valeurs.. 18, 19, 29, 31, 36, 39, 40, 46, 55, 61, 62

Expression régulière Une expression régulière, ou expression rationnelle est une chaîne de caractère, écrite selon une syntaxe précise, qui permet de détecter des motifs dans du texte [13]. Les expressions régulières s'appuient sur la classification des caractères en classes (minuscules, majuscules, chiffres, espaces), sur des structures alternatives (un caractère ou un autre) et exclusives (un caractère n'ayant pas certaines propriétés) pour repérer des motifs. Par exemple, « 2022 » correspond au motif : `\d4`, soit « quatre chiffres à la suite ». Une adresse mail est également un motif : `[^(@|\s)]+@[^(@|\s)]+`, soit « plusieurs caractères qui ne sont ni un espace une arobase, suivi d'une arobase, suivi de plusieurs caractères qui n'est ni un espace ni une arobase ».. 14, 18, 27, 31, 42, 46, 51, 52, 89

Fichier de log Un fichier de log est un fichier qui est créé pendant l'exécution d'un algorithme pour sauvegarder des informations qui le concernent. Il s'agit en général d'informations annexes sur l'exécution du script : messages d'erreur, données produites en cours d'utilisation.... 61, 63, 67–69, 76, 78, 113

HTTP Le *Hypertext Transfer Protocol* (HTTP, ou P« protocole de transfert hypertexte »), est un protocole qui permet la communication entre différentes machines dans une architecture client-serveur, comme celle du Web. C'est donc le protocole qui est à la base d'Internet, puisqu'il établit la manière dont des machines peuvent interagir au sein d'un réseau. Il définit un ensemble de méthodes pour interagir avec le serveur – demander (`GET`) des données, en envoyer (`POST`)... Il définit également la manière dont le serveur répond au client et un ensemble d'erreurs [14], chacune disposant d'un code pour l'identifier (l'erreur 404 indiquant que la ressource demandée n'existe pas, ou l'erreur 418 signifiant « Je suis une théière », un canular qui pouve l'humour des concepteur.ice.s du Web).. 58

Named Entity Linking (NIL) Le *Named Entity Linking*, ou « liage d'entités nommées » consiste à lier des entités – c'est-à-dire des objets uniques, comme des lieux, des personnes ou des organisations – à des sources de connaissance externes, comme des bases de connaissance en ligne.. 69

Python Python est un language de programmation impérative (c'est-à-dire, basé sur la production et la transformation de données par une suite d'instructions) créé en

13. **noauthor_expression_2022**.
14. **fielding_architectural_2000**.

1991. Il est particulièrement utilisé dans le domaine du traitement de données et des humanités numériques. C'est le langage le plus utilisé dans le projet *MSS / Katabase*. 36

Score BLEU Un score BLEU (bilingual evaluation understudy) est un score servant à évaluer la qualité de la traduction d'un texte par une machine. Il est obtenu en comparant le texte traduit par une machine avec un plusieurs textes de référence (c'est-à-dire, des traductions faites par des humains du même texte).. 90

Score F1 Le score F1, ou *F-score*, est la moyenne harmonique de la précision (vrais positifs par rapport au total de résultats obtenus) et du rappel (nombre de résultats positifs par rapport au total de résultats positifs). [15]. Le score F1 a l'avantage de prendre en compte les vrais et les faux positifs. Ce score, dont la valeur est contenue entre 0 et 1, permet de mesurer l'exactitude d'un algorithme d'apprentissage machine, ou d'un moteur de recherche. Il se calcule de la manière suivante :

$$f = 2 \times \frac{\text{précision} \times \text{rappel}}{\text{précision} + \text{rappel}}$$

. 14, 19, 69, 70, 113

SPARQL SPARQL est un language de requête qui permet d'interagir avec une base de données au format RDF. « SPARQL exprime des requêtes sur des sources de données diverses [...]. SPARQL rend possible la requête de données en graphes [...], avec des données conjointes et disjointes. SPARQL permet également l'agrégation de données, les sous-requêtes, la négation, la création de données à l'aide d'expression, le test de données et la contrainte des requêtes. Les résultats de requêtes sparql peuvent être des jeux de résultats ou des graphes RDF. » [16]. 13, 14, 18–20, 56, 71–80, 95, 101, 107

Table de conversion Une table de conversion est, tout simplement, un dictionnaire qui contient en clés un certain nombre d'informations telles qu'elles figurent dans le texte et, en valeurs, une version normalisée de cette information. Cela permet de détectier des motifs à extraire autant de normaliser les informations.. 19, 27, 31

URI Une URI (*Uniform Resource Identifier*) est une chaîne de caractères qui identifie une ressoource sur un réseau. Cette URI est permanente, et permettra d'identifier une ressource même si celle-ci est supprimée. L'URL est un type particulier d'URI

15. **noauthor_precision_2022** ; **noauthor_f-score_2022**.

16. « SPARQL can be used to express queries across diverse data sources [...]. SPARQL contains capabilities for querying [...] graph patterns along with their conjunctions and disjunctions. SPARQL also supports aggregation, subqueries, negation, creating values by expressions, extensible value testing, and constraining queries by source RDF graph. The results of SPARQL queries can be result sets or RDF graphs. »**harris_sparql_2013** (traduction de l'auteur).

qui, en plus de d"crire une resource, la représente aussi : un URL de page *Wikipedia* permet d'identifier une page de façon unique, comme toute URI (aucune autre page sur internet n'ayant le même URL) ; mais en plus, elle pêermet d'accéder à la page elle-même.. 79

URL Une URL (*Uniform Resource Locator*) est un identifiant unique sous la forme d'une chaîne de caractères pointant vers une ressource sur le Web. De la même manière qu'une cote de bibliothèque pointe vers un livre précis, un URL pointe vers une page ou une donnée sur le Web. L'URL permet d'agir sur une ressource (en envoyant des données...) et peut être associée à des données ainsi qu'à des reésentations (une page Web, par exemple).. 60

Acronymes

API Application Programming Interface. 14, 19, 21, 23, 24, 26, 29, 32, 37, 45, 48, 49, 51, 53, 56–63, 65–67, 69, 101, 106, 139, *Glossaire :* API

HTTP *Hypertext Transfer Protocol*. 58–60, 77, *Glossaire :* HTTP

NIL Named Entity Linking (NIL). 69, 70, 103, 115, *Glossaire :* Named Entity Linking (NIL)

URI *Uniform Resource Identifier*. 79, *Glossaire :* URI

URL *Uniform Resource Locator*. 60, 61, 63, *Glossaire :* URL

Table des figures

Liste des tableaux

Table des matières

Annexe A

Lien vers tout le code source de la chaîne de traitement *Katabase*

L'intégralité du code produit pour le projet *Katabase*, depuis la création des documents `TEI` jusqu'à l'application Web, sont disponibles sur la plateforme d'hébergement de code GitHub.

— Plateforme du projet

— Première étape : normalisation des documents `TEI`

— Deuxième étape : extraction de données et normalisation

— Troisième étape : enrichissement de données à l'aide de *Wikidata*

— Quatrième étape : création de jeux de données `JSON` pour le site web

— Application web *Katabase*

L'intégralité du code source est disponible sous licence libre (GNU GPL v3.0, MIT ou Creative Commons). Le site web peut également être consulté en ligne à cette adresse.

Annexe B

Résultat des tests de l'algorithme d'extraction d'informations de *Wikidata*

Les résultats des tests présentés dans les tables ci-dessous (B.1, B.2) ont été menés sur un jeu de 200 couples `name` et `trait`. Ils ont été choisis de façon à être représentatifs du jeu de données complet, avec une variété d'entrées semblables (personnes nobles et non nobles, entrées qui ne sont pas consacrées à des personnes...). Ce jeu de test présente également une structure et un degré de détail semblable au jeu de données complet. Il y a notamment exactement la même proportion d'entrées présentant un `trait` que dans le jeu de test et le jeu final.

Les résultats en pourcentage sont toujours exprimés en **pourcentages du nombre total d'entrées requêtées** (soit 200).

Type de requête	**Noms**	**Noms et nom de famille noble**	**Noms et titre de noblesse**	**Noms et date de naissance et de mort**	**Noms et occupation**
Avec reconstitution des prénoms	48%	45.9%	50%	52.6%	53.1%
Sans reconstitution des prénoms	42%	40.5%	50%	52.6%	53.1%

TABLE B.1 – Tests menés avec un jeu de 200 entrées sur des paramètres isolés

Le tableau ci-dessus (B.1) présente les résultats de tests menés pour isoler l'impact de chaque paramètee dans l'obtention du bon résultat. Les mêmes requêtes ont été lancées sur un jeu de test de 200 entrées de catalogue. Toutes les requêtes contient le prénom et le nom de famille complets (« Noms » dans la table). La première requête ne contient que ce paramètre, tandis que toutes les autres requêtes sont lancées avec un paramètre en plus : titre de noblesse, nom de famille noble, dates et occupation. Le fait que ces paramètres soient utilisés ne garantit pas qu'ils soient disponibles dans le jeu de données utilisé en entrée. Ce test permet d'isoler de mieux construire l'algorithme final de recherches en plein texte sur le moteur de recherche de *Wikidata*, puisqu'il permet de voir quels sont les paramètres les plus fiables.

Bons identifiants récupérés	65%
Score F1	0.674
Précision	0.677
Rappel	0.670
Résultats certains	36%
Faux positifs parmi les éléments certains	8.5%
Total d'entrées requêtées	200
Identifiants récupérés	192
Silence	8
Temps d'exécution avec l'option `fetch`	88.3 secondes
Temps d'exécution sans l'option `fetch`	92.49 secondes

TABLE B.2 – Tests menés avec un jeu de 200 entrées sur l'algorithme final

Le tableau ci-dessus présente les résultats d'un test évaluant les performances de l'algorithme final. Voici une explication de ces résultats :

— *Bons identifiants récupérés* : la proportion, exprimée en pourcentage, du nombre d'identifiants récupérés par l'algorithme qui correspondent aux identifiants récupérés manuellement

— *Score F1* : le score F1, soit la moyenne pondérée de la précision et du rappel. Contrairement à la mesure ci-dessus, le score F1 prend en compte le silence et les faux négatifs, et offre donc une interprétation plus complète de la qualité du script.

— *Précision* : la précision est une composante du score F1 qui mesure la proportion d'identifiants pertinents obtenus parmi tous les identifiants obtenus.

— *Rappel* : le rappel est une composante du score F1 qui mesure la proportion d'identifiants pertinents obtenus parmi l'ensemble des identifiants pertinents qui existent.

— *Résultats certains* : pour accélérer le processus de relecture, un score de certitude a été attribué à certains identifiants obtenus grâce à l'algorithme. Ce score dépend du nombre de paramètres utilisés dans la recherche qui a permis de récupérer l'identifiant. 36% des résultats obtenus sont considérés comme certains.

— *Faux positifs parmi les éléments certains* : le score de certitude présenté ci-dessus admet un taux d'erreur qui est ici quantifié : 8.5% des résultats obtenus sont erronnés alors qu'ils ont été considérés comme certains (8,5% du jeu de données total, donc).

— *Total d'entrées requêtées* : le volume du jeu de données de test, soit 200 entrées.

— *Identifiants récupérés* : le nombre d'identifiants obtenus, qu'ils soient corrects ou non.

— *Silence* : le nombre d'entrées pour lesquelles aucun identifiant n'a été obtenu.

— *Temps d'exécution avec l'option `fetch`* : le temps pris, en secondes, pour lancer l'intégralité de l'algorithme de récupération d'identifiants *Wikidata* sur les 200 entrées en stockant dans des fichiers de log les requêtes lancées et les identifiants requêtés.

— *Temps d'exécution sans l'option* `fetch` : le temps pris, en secondes, pour lancer l'intégralité de l'algorithme de récupération d'identifiants sur les 200 entrées sans utiliser de fichiers de log.

Base de connaissances / Score	Score F1 – recherche des candidates	Score F1 – Lien avec le candidat	Overall linking accuracy
DBPedia	0.899	0.536	0.834
DataBNF	0.689	0.726	0.7
Wikidata	0.869	0.611	0.85

TABLE B.3 – Résultats du NIL de lieux dans des textes littéraires du XIXe s. par Soudani et al. (2018)

Scores obtenus par Aicha Soudani et al. pour le liage d'entités nommées pour des lieux dans des textes littéraires français du XIX$^{\text{ème}}$ s., présentés à la conférence *HumaNS'2018*[1]. Les scores F1 ne figurent pas dans l'article original, qui ne présente que la précision et le rappel. Je les ai donc calculés moi même. La méthode de calcul de l'*overall linking accuracy* (« exactitude générale du liage ») n'est pas précisée dans l'article ; il est seulement indiqué que « Cette mesure essaie d'évaluer l'efficacité globale du système, et non par phase, et exprime la fiabilité de REDEN pour la tâche de résolution des entités nommées. »[2].

La méthode utilisée dans cet article est la suivante : les auteur.ice.s s'appuient fortement sur l'apprentissage machine, en identifiant d'abord des entités dans le texte à l'aide de `SEM` avant de lier les entités à différentes bases de connaissance en utilisant `REDEN`.

1. **soudani_adaptation_2018**.
2. **soudani_adaptation_2018**.

Annexe C

Graphiques

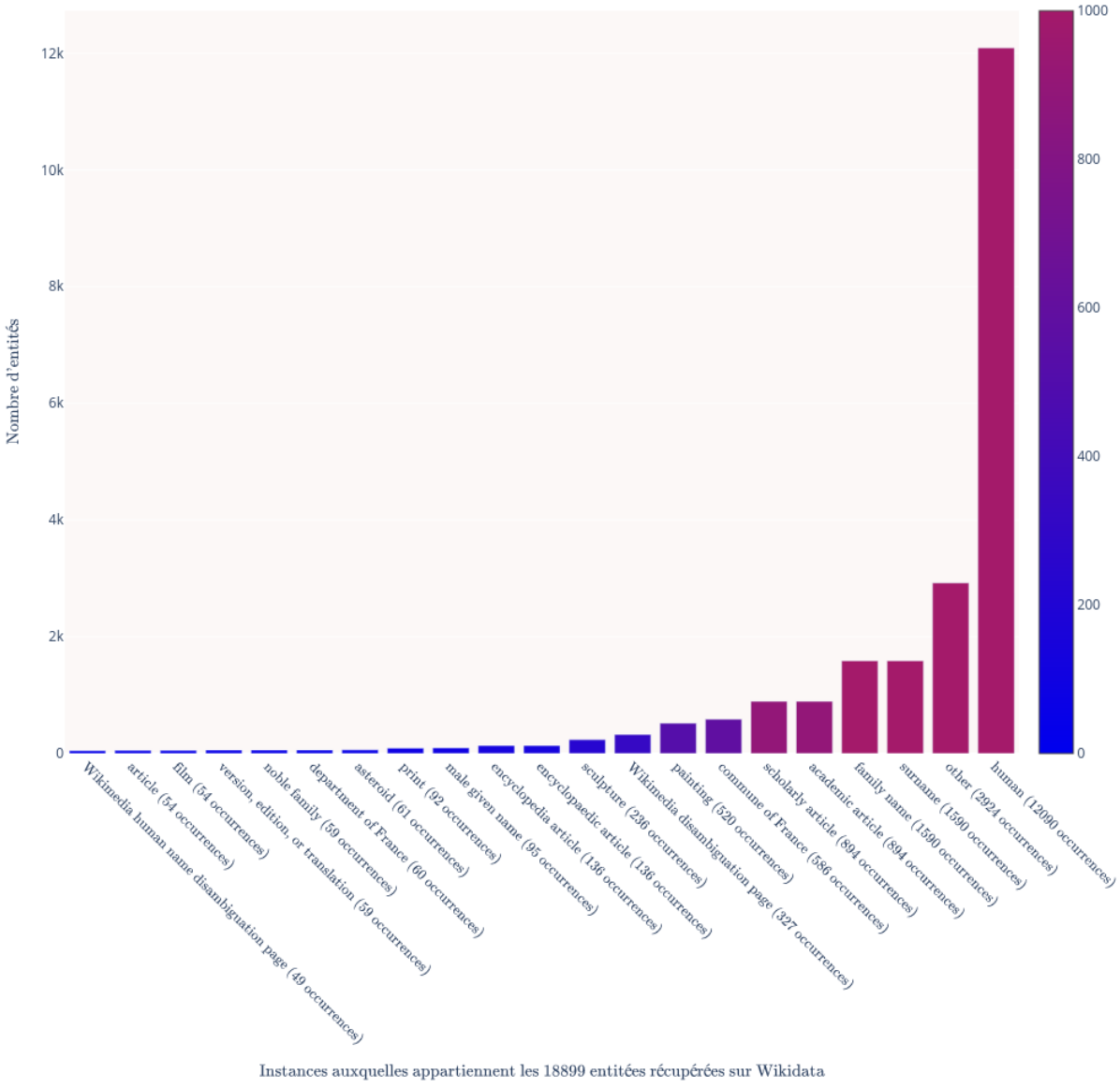


FIGURE C.1 –
Occurrences des différentes catégories auxquelles appartiennent les entités *Wikidata* liées avec les entrées de catalogues

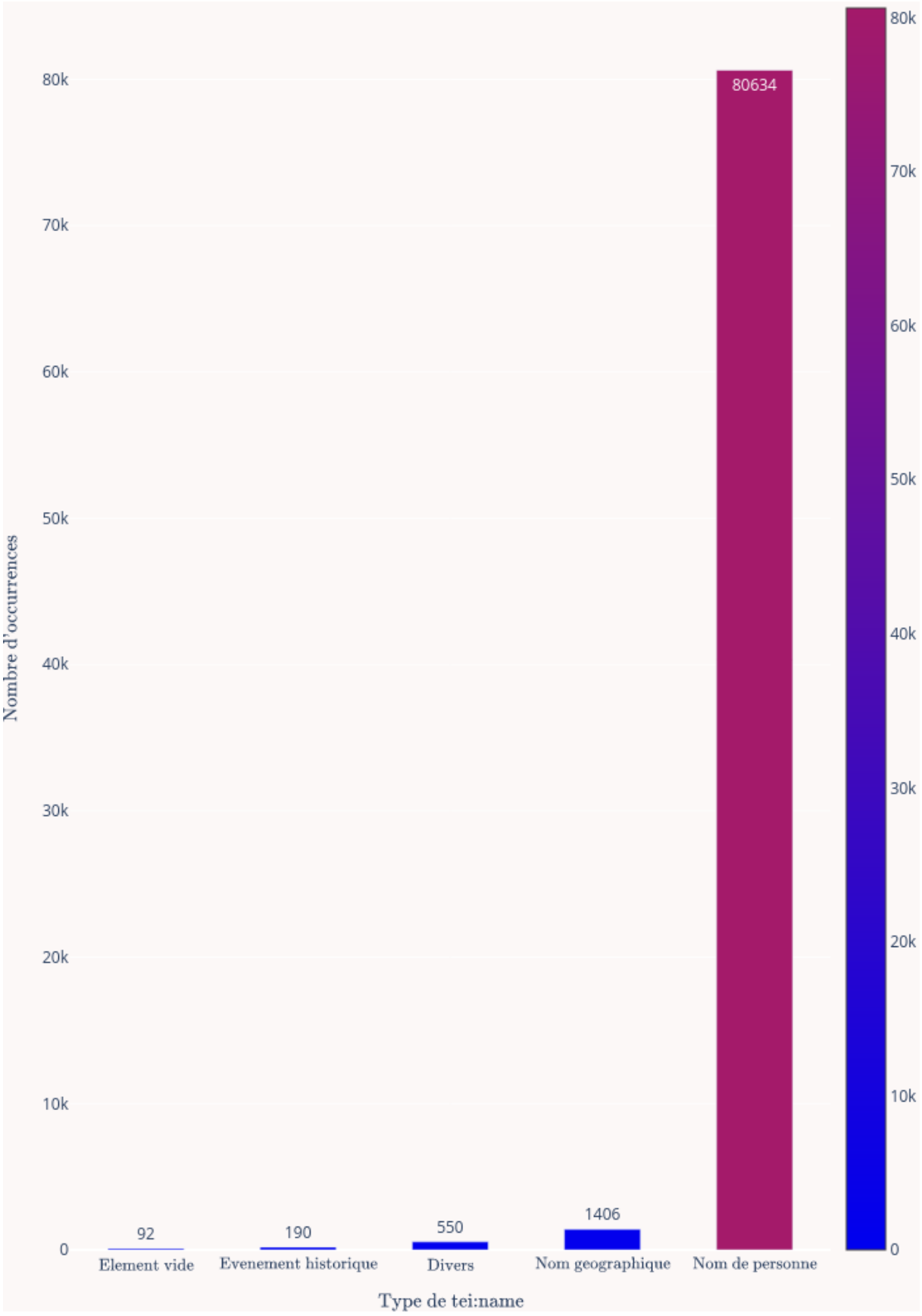


FIGURE C.2 – Répartition des différents types de `name`

Annexe D

Code source et données encodées

```
functions = {
    "général": "general",
    "maréchal": "marshal",
    "lieutenant": "military",
    "officier": "military",
    "colonel": "military",
    "lieutenant-colonel": "military",
    "commandant": "military",
    "capitaine": "military",  # "less important" military positions
    "roi": "king",
    "empereur": "emperor",
    "president": "president",
    "homme politique": "politician",
    "président de l'assemblée": "politician",
    "orateur": "politician",
    "député": "politician",
    "secrétaire d'état": "politician",
    "sénateur": "politician",
    "écrivain": "writer",
    "auteur": "writer",
    "romancier": "writer",
    "acteur": "actor",
    "actrice": "actress",
    "cantatrice": "singer",
    "chanteur": "singer",
    "chanteuse": "singer",
    "peintre": "painter",
    "sculpteur": "sculptor",
    "statutaire": "sculptor",
    "compositeur": "composer",
    "musicien": "musician",
    "musicienne": "musician",
    "tragédien": "actor",
    "chansonnier": "chansonnier",
    "achitecte": "architect",
    "journaliste": "journalist",
    "inventeur": "inventor",
    "chimiste": "chemist",
    "connétable": "constable",
    "archevêque": "archbishop",
    "évêque": "bishop",
    "docteur": "physicist",
    "médecin": "physicist"
}
```

Code source D.1 – Table de conversion associant un métier à son équivalent normalisé

```
dpts = [
    "ain",
    "aisne",
    "allier",
    "basses-alpes",
    "hautes-alpes",
    "alpes-maritimes",
    "annepins",
    "provence",
    "ardèche",
    "ardennes",
    "arriège",
    "arno",
    "aube",
    "aude",
    "aveyron",
    "bouches-de-l'elbe",
    "bouches-de-l'escaut",
    "bouches-de-l'yssel",
    "bpuches-de-la-meuse",
    "bouches-du-rhin",
    "bouches-du-rhône",
    "bouches-du-weser",
    "calvados",
    "cantal",
    "charente",
    "charente-inférieure",
    "cher",
    "corrèze",
    "corse",
    "côte-d'or",
    "côtes-du-nord",
    "creuse",
    "deux-nèthes",
    "deux-sèvres",
    "doire",
    "dordogne",
```

```
    "doubs",
    "drôme",
    "dyle",
    "ems-occidental",
    "ems-oriental",
    "ems-supérieur",
    "escaut",
    "eure",
    "eure-et-loir",
    "finistère",
    "forêts",
    "gard",
    "haute-garonne",
    "gers",
    "gironde",
    "hérault",
    "ille-et-villaine",
    "indre",
    "indre-et-loire",
    "isère",
    "jemappes",
    "jura",
    "landes",
    "léman",
    "loire",
    "loir-et-cher",
    "haute-loire",
    "loire-inférieure",
    "loiret",
    "lot",
    "lot-et-garonne",
    "lozère",
    "lys",
    "maine-et-loire",
    "manche",
    "marengo",
    "marne",
    "haute-marne",
```

```
    "méditerrannée",
    "mayenne",
    "meurthe",
    "meuse",
    "meuse-inférieure",
    "mont-blanc",
    "mont-tonnerre",
    "montenotte",
    "morbihan",
    "meuse",
    "moselle",
    "nièvre",
    "nord",
    "oise",
    "ombrone",
    "orne",
    "ourte",
    "paris",
    "pas-de-calais",
    "pô",
    "puy-de-dôme",
    "hautes-pyrénées",
    "basses-pyrénées",
    "pyrénées-orientales",
    "haut-rhin",
    "bas-rhin",
    "rhin-et-moselle",
    "rhône",
    "rhône-et-loire",
    "roer",
    "rome",
    "haute-saône",
    "saône-et-loire",
    "sambre-et-meuse",
    "sarre",
    "sarthe",
    "seine",
    "seine-et-marne",
```

```
    "seine-et-oise",
    "seine-inférieure",
    "sézia",
    "simplon",
    "deux-sèvres",
    "somme",
    "stura",
    "tarn",
    "tarn-et-garonne",
    "taro",
    "trasimène",
    "var",
    "vaucluse",
    "vendée",
    "vienne",
    "haute-vienne",
    "vosges",
    "yonne",
    "yssel-supérieur",
    "zuyderzée"
]
```

Code source D.2 – Liste de départements du XIXe s. pour détecter des informations géographiques

```
countries = {
    "états-unis d'amérique": "united states of america",
    "etats-unis d'amérique": "united states of america",
    "états unis d'amérique": "united states of america",
    "etats unis d'amerique": "united states of america",
    "états-unis": "united states of america",
    "etats-unis": "united states of america",
    "etats unis": "united states of america",
    "états unis": "united states of america",
    "italie": "italy",
    "grèce": "greece",
    "canada": "canada",
    "chine": "china",
    "haïti": "haiti",
    "tobago": "tobago",
    "brésil": "brasil",
    "burkina-faso": "burkina-faso",
    "cameroun": "cameroun",
    "tchad": "tchad",
    "congo": "congo",
    "gabon": "gabon",
    "guinée": "guinea",
    "côte d'ivoire": "ivory coast",
    "mali": "mali",
    "mauritanie": "mauritania",
    "niger": "niger",
    "sénégal": "senegal",
    "madagascar": "madagascar",
    "seychelles": "seychelles",
    "tanzanie": "tanzania",
    "zanzibar": "zanzibar",
    "liban": "lebanon",
    "syrie": "syria",
    "inde": "india",
    "laos": "laos",
    "viet-nâm": "vietnam"
}
```

Code source D.3 – Table de conversion pour les pays

```
colonies = [
    "québec",
    "ontario",
    "saint-pierre-et-miquelon",
    "mississippi",
    "missouri",
    "louisiane",
    "anguilla",
    "antigua",
    "dominique",
    "saint-domingue",
    "guadeloupe",
    "monsterrat",
    "saint-martin",
    "saint-barthélémy",
    "sainte-lucy",
    "saint-vincent-et-les-grenadines",
    "saint-eustache",
    "saint-christophe",
    "martinique"
    "guyane française",
    "guyane",
    "maroc",  # unfortunately the morocco referred to in XIXth century
    ↪ france is a french protectorate
    "algérie",  # same
    "algérie française",  # same
    "tunisie",  # same
    "fezzan",
    "dahomey",
    "haute-volta",
    "oubangui-chari",
    "congo français",
    "moyen-congo",
    "guinée française",
    "soudan français",
    "gorée",
    "tigi",
```

```
    "djibouti",
    "cheikh saïd",
    "comores",
    "fort-dauphin",
    "îles maurice",
    "mayotte",
    "la réunion",
    "îles éparses",
    "île amsterdam",
    "île saint-paul",
    "archipel crozet",
    "îles kerguelen",
    "castellorizo",
    "grand-liban",
    "sandjak d'alexandrette",
    "indes françaises",
    "pondichéry",
    "karikal",
    "yanaon",
    "mahé",
    "chanderngor",
    "tonkin",
    "annam",
    "cochinchine",
    "guangzhou wan",
    "shanghai",
    "guangzhou",
    "tianjin",
    "hankou",
    "clipperton",
    "nouvelle-calédonie",
    "polynésie française",
    "vanuatu",
    "nouvelles-hébrides",
    "wallis et futuna"
]
```

Code source D.4 – Liste d'anciennes colonies françaises utilisées pour la détection de motifs

```
provinces = [
    "armagnac",
    "île-de-france",
    "berry",
    "orléanais",
    "normandie",
    "languedoc",
    "lyonnais",
    "dauphiné",
    "champagne",
    "aunis",
    "saintonge",
    "poitou",
    "guyenne et gascogne",
    "bourgogne",
    "picardie",
    "anjou",
    "provence",
    "angoumois",
    "bourbonnais",
    "marche",
    "bretagne",
    "maine",
    "touraine",
    "limousin",
    "comté de foix",
    "auvergne",
    "béarn",
    "alsace",
    "artois",
    "roussillon",
    "flandre française et hainaut français",
    "franche-comté",
    "lorraine et trois-évêchés",
    "corse",
    "nivernais",
]
```

Code source D.5 – Liste d'anciennes provinces françaises pour la détection de motifs

```
events = {
    "défense nationale": "government of national defense",
    "defense nationale": "government of national defense",
    "révolution française": "french revolution",
    "revolution francaise": "french revolution",
    "guerre de trente ans": "thirty years' war 1618 1648",
    "guerre de cent ans": "hundred years' war 1337 1453",
    "guerre de sept ans": "seven years war 1756 1763",
    "guerre": "war",
    "insurrection": "war",
    "siège de mayence": "siege of mainz",
    "siège": "siege",
    "commune": "commune",
    "défense": "battle",
    "révolution": "revolution"
}
```

Code source D.6 – Table de conversion pour les évènements historiques

```
status = {
    "empereur": "",
    "impératrice": "",
    "général": "general",
    "reine": "queen",
    "roi": "king",
    "princesse": "princess",
    "prince": "prince",
    "archiduchesse": "",
    "archiduc": "",
    "duchesse": "duchess",
    "duc": "duke",
    "famille": "family",
    "seigneur": "",
    "vicomtesse": "",
    "victesse": "",
    "vicomte": "",
    "victe": "",
    "comtesse palatine": "countess palatine",
    "comtesse": "",
    "ctesse": "",
    "comte": "",
    "cte": "",
    "cardinal": "",
    "pape": "pope",
    "lord": "",
    "chevalier": "",
    "marquise": "",
    "marquis": "",
    "sire": "",
    "baronnesse": "",
    "baronne": "",
    "baron": "",
    "abbé": "",
    "madame": "",
    "mme": "",
    "monsieur": "",
    "mr": "",
    "docteur": "",
    "maréchale": "",
    "maréchal": "",
    "mademoiselle": "",
    "melle": "",
    "mlle": "",
    "sir": ""
}
```

Code source D.7 – Liste de colonies pour la détection de motifs

```
comp_names = {
    "arm ch": "armand-charles",
    "ch m": "charles-marie",
    "ch l f": "charles-louis-françois",
    "f m": "francois-marie",
    "fr emm.": "françois-emmanuel",
    "j ant": "jean-antoine",
    "j f": "jean-francois",
    "j m": "jean-marie",
    "j j": "jean-jacques",
    "j l": "jean-louis",
    "j b": "jean-baptiste",
    "j p": "jean-pierre",
    "j pierre": "jean-pierre",
    "l f": "louis-françois",
    "m f": "marius-felix",
    "franc rené": "francois-rené",
    "m madeleine": "marie-madeleine",
    "ph h": "philippe henri",
    "p aug": "pierre auguste",
    "p alex": "pierre alexandre",
    "p j": "pierre-jean",
    "j sylvain": "jean-sylvain",
    "l ph": "louis-philippe",
    "edm ch": "edmond-charles",
    "ch marie": "charles-marie"
}
```

Code source D.8 – Table de conversion permettant de remplacer un nom abrégé composé par sa version complète

```
names = {
    "ad": "adam",
    "alex": "alexandre",
    "alph": "alphonse",
    "ant": "antoine",
    "arm": "armand",
    "aug": "auguste",
    "ch": "charles",
    "cl": "claude",
    "dom": "dominique",
    "emm": "emmanuel",
    "ed": "edouard",
    "et": "etienne",
    "ét": "etienne",
    "ferd": "ferdinand",
    "fred": "frederic",
    "fr": "françois",
    "franc": "françois",
    "franç": "françois",
    "fréd": "frédéric",
    "g": "guillaume",
    "guill": "guillaume",
    "gab": "gabriel",
    "jh": "joseph",
    "jacq": "jacques",
    "jos": "joseph",
    "math": "matthieu",
    "nic": "nicolas",
    "ph": "philippe",
    "v": "victor",
    "vr": "victor",
}
```

Code source D.9 – Table de conversion permettant de remplacer un nom abrégé non composé par sa version complète

```
def rgx_abvcomp(nstr):
    """
    try to extract an abbreviated composed first name. if there is no match, return None
    pattern
    -------
    the patterns in the example below are simplified to keep things readable
    - two strings separated by a "-" or "\s"
    - the first or second string can be a full name ([A-Z][a-z]+) or an abbreviation ([A-Z][a-z]*\.)
    - if the strings are separated by "\s", they must be finished by "\."
    (to be sure that we don't capture full names, i.e: "J. Ch."  can be captured,
    but not "Jean Charles")
    - complex names with 3 or more words must have "-" and at least one "\."
    - (\s|$) and (^|\s) are safeguards to avoid matching the end or beginning of another word
    examples
    --------
    matched : M.-Madeleine Pioche de la Vergne  # matched string : M.-Madeleine
    matched : C.-A. de Ferriol  # matched string : C.-A.
    matched : J. F.  # matched string : J. F.
    matched : Jean F.  # matched string : Jean F.
    matched : Jean-F.  # matched string : Jean-F.
    matched : A M  # matched string : A M
    matched : C.-Edm.-G.  # matched string : C.-Edm.-G.
    matched : Charles-Edm.-G.  # matched string : Charles-Edm.-G.
    not matched : Anne M
    not matched : Claude Henri blabla
    not matched : Claude Henri
    :param nstr: the name string used as input
    :return: the matched string if there is a match ; None if there is no match
```

```
    """
    mo = re.search(r"(^|,|\s)[A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ][a-zàáâäéèêëíìîïòóôöúùûüøœæç]*"
            + "\.?-[A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ][a-zàáâäéèêëíìîïòóôöúùûüøœæç]*\.(\s|,|$)", nstr)
         or re.search(r"(^|,|\s)[A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ][a-zàáâäéèêëíìîïòóôöúùûüøœæç]*\."
            + "-[A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ][a-zàáâäéèêëíìîïòóôöúùûüøœæç]*\.?(\s|,|$)", nstr)
         or re.search(r"(^|,|\s)[A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ]\.?\s"
            + "[A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ][a-zàáâäéèêëíìîïòóôöúùûüøœæç]*\.(\s|,|$)", nstr)
         or re.search(r"(^|,|\s)[A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ][a-zàáâäéèêëíìîïòóôöúùûüøœæç]*\.?"
            + "\s[A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ]\.(\s|,|$)", nstr) \
         or re.search(r"(^|,|\s)[A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ]\.?\s[A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ]\.?(\s|,|$)", nstr)
         \
         or re.search(r"([A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ]\.){2,}", nstr) \
         or re.search(r"(^|,|\s)([A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ][a-zàáâäéèêëíìîïòóôöúùûüøœæç]*\.?-)+"
            + "([A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ][a-zàáâäéèêëíìîïòóôöúùûüøœæç]*\.)(\s|,|$)", nstr)
         \
         or re.search(r"(^|,|\s)([A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ][a-zàáâäéèêëíìîïòóôöúùûüøœæç]*\.-)+"
             + "([A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ][a-zàáâäéèêëíìîïòóôöúùûüøœæç]*\.?)(\s|,|$)",
             nstr)
    if mo is not None:
        return mo[0]
    else:
        return None
```

Code source D.10 – Fonction permettant d'identifier et d'extraire un nom abrégé composé

```python
def rgx_abvsimp(nstr):
    """
    try to extract a "simple" (not composed) abbreviated first name. if there is no match, return None
    pattern
    -------
    a capital letter (possibly followed by a certain number of lowercase letters)
    ended with a dot. (\s|$) and (^|\s) are safeguards to avoid matching the beginning
    end of another word.
    *warning* : it can also capture parts of composed abbreviated names => must be used
    in an if-elif after trying to match a composed abbreviated name
    examples
    --------
    matched : bonjour Ad.  # matched string : Ad.
    matched : J. baronne  # matched string : J.
    matched : J. F.  # matched string : J.
    matched : Jean F.  # matched string : F.
    not matched : A.-M.
    not matched : Anne M
    not matched : Hector
    :param nstr: the name string used as input
    :return:
    """
    mo = re.search(r"(^|\s)[A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ][a-zàáâäéèêëíìîïòóôöúùûüøœæç]*\.(\s|$|,)", nstr)
    if mo is not None:
        return mo[0]
    else:
        return None
```

Code source D.11 – Fonction permettant de repérer et d'extraire un nom abrégé simple

```
def rgx_complnm(nstr):
    """
    try to extract a complete name from a string. if there is noœ
↪   match, return None
    pattern
    -------
    - an uppercase letter followed by several lowercase letters ;
    - this pattern can be repeated several times, separated by a space
↪   or "-"
    - (\s|$) and (^|\s) are safeguards to avoid matching the beginning
↪   or end of another word.
    :param nstr: the string from which a name should be extracted
    :return:
    """
    mo = re.search(r"(^|\s)[A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ][a-zàáâäéèêëíìîïòóôöúùûüø⌋
↪   œæç]+"
        + "((\s|-)[A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ][a-zàáâäéèêëíìîïòóôöúùûüøœæç]+)*($⌋
↪   |\s|,)",
↪   nstr)
    if mo is not None:
        return mo[0]
    else:
        return None
```

Code source D.12 – Fonction permettant d'identifier et d'extraire un nom complet non abrégé

```xml
<item n="264" xml:id="CAT_000327_e264">
    <!-- ... -->
    <name type="author">Spontini (Gaspard)</name>
    <trait>
        <p>célèbre compositeur, auteur de la Vestale, né en 1779, mort
        ↪ en 1851.</p>
    </trait>
    <!-- ... -->
</item>
```

(a) La source XML-TEI

```python
{
    'fname': 'gaspard ',
    'lname': 'spontini ',
    'nobname_sts': '',
    'status': '',
    'dates': '1779 1851 ',
    'function': 'writer',
    'rebuilt': False
}
```

(b) Le dictionnaire d'informations structurées

```
gaspard spontini 1779 1851 writer
gaspard spontini 1851 writer
gaspard spontini 1779 writer
gaspard spontini writer
spontini 1779 1851 writer
spontini 1851 writer
spontini 1779 writer
spontini writer
```

(c) Les recherches lancées sur l'API *Wikidata*

FIGURE D.1 – De la TEI à l'API : document en entrée, informations extraites et chaînes de caractères recherchées

Ci dessus (D.1) sont présentés l'entrée et les données produites lors de l'alignement avec *Wikidata* : en premier les données en entrée, ensuite les informations qui en sont extraites et enfin les différentes chaînes de caractères recherchées. Il est à noter que, dans la plupart des cas, seules une ou deux recherches sont faites sur l'API avant d'obtenir un résultat ; ici, huit chaînes différentes ont été construites et recherchées, ce qui représente l'intégralité de l'algorithme.

Annexe E

Images

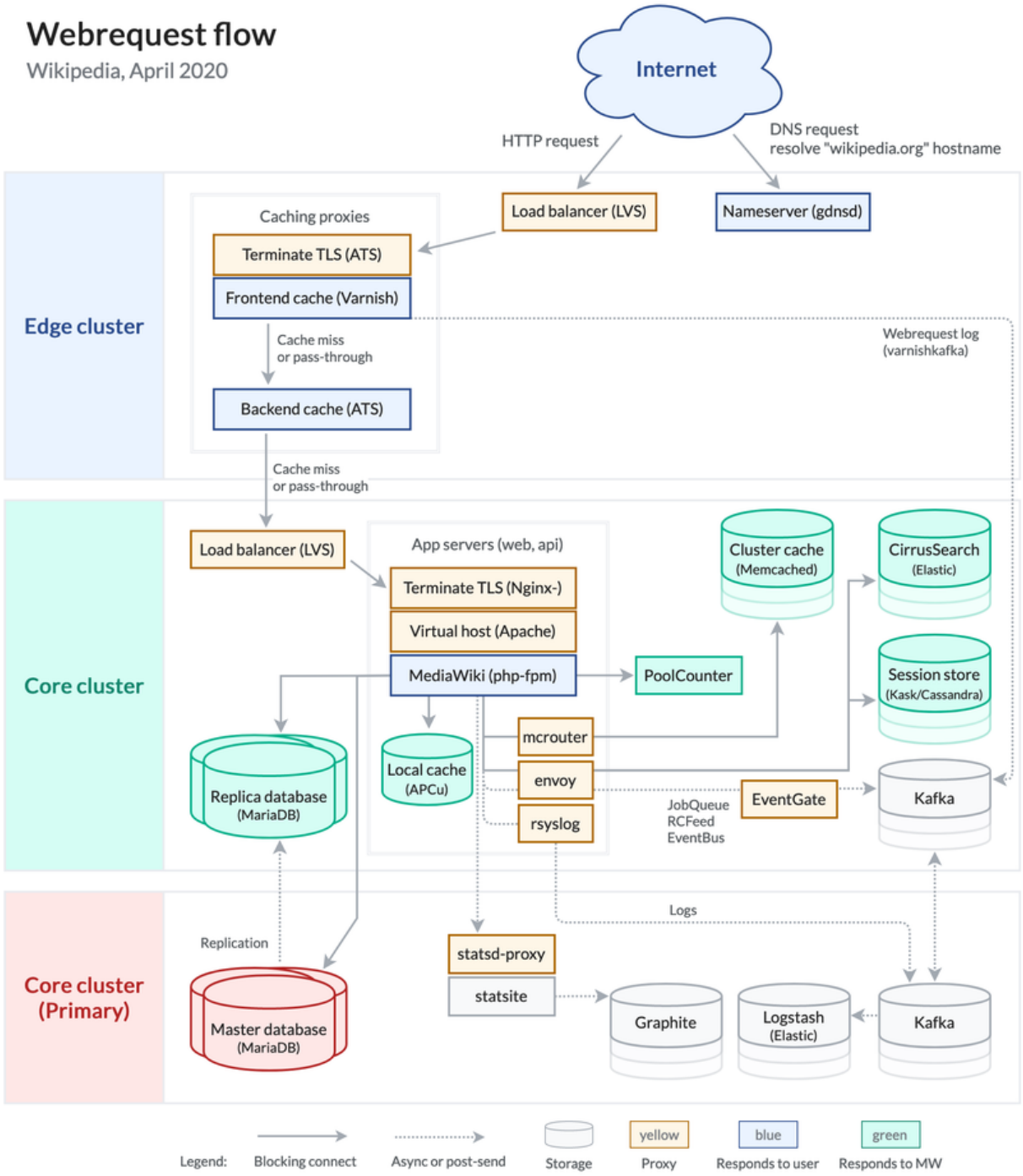


FIGURE E.1 – L'architecture des bases de données de la *Wikimedia Foundation* (schéma réalisé par Timo Tijhof en 2020, disponible dans le domaine public avec la licence Creative Commons CC0.1.0 Universal Public Domain License)